Embedded Early Vision Techniques for Efficient Background Modeling and Midground Detection

A Dissertation

Presented to

The Academic Faculty

by

Brian Valentine

In Partial Fulfillment

Of the Requirements for the Degree

Doctor of Philosophy in the

School of Electrical Engineering

Georgia Institute of Technology

May 2010

Embedded Early Vision Techniques for Efficient Background Modeling and Midground Detection

Approved by:

Dr. Linda M. Wills, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. D. Scott Wills, Co-advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. George Riley
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Patricio Vela
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Aaron Lanterman
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Milos Prvulovic
College of Computing
Georgia Institute of Technology

Date Approved : 18th March 2010

ACKNOWLEDGEMENTS

I would first like to thank Dr. Linda Wills, my advisor, for her guidance throughout my time at Georgia Tech. Her insight and critical analyses has allowed me to develop research and writing skills that I will use throughout my career. I would also like to thank my co-advisor, Dr. Scott Wills, whose energy and thirst for knowledge always inspired me to think deeper on research topics and find ways to innovate. I am grateful for many of the insightful discussions and career guidance I have received from them.

I am thankful to Dr. George Riley, Dr. Patricio Vela, Dr. Aaron Lanterman, and Dr. Milos Prvulovic for serving on my committee and giving their feedback on my research and presentation skills. I would also like to thank Dr. Senyo Apewokin, Michael Bales, Jee Choi, and Dana Forsthoefel for the collaborative group discussions, feedback on papers, and work on joint publishing efforts. Finally, I would like to thank my family, whose unwavering support throughout this educational journey has enabled me to achieve my highest aspirations.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

SUMMARY

An automated vision system performs critical tasks in video surveillance, while decreasing costs and increasing efficiency. It can provide high quality scene monitoring without the limitations of human distraction and fatigue. Advances in embedded processors, wireless networks, and imager technology have enabled computer vision systems to be deployed pervasively in stationary surveillance monitors, hand-held devices, and vehicular sensors. However, the size, weight, power, and cost requirements of these platforms present a great challenge in developing real-time systems. This dissertation explores the development of background modeling algorithms for surveillance on embedded platforms. Our contributions are as follows:

- An efficient pixel-based adaptive background model, called *multimodal mean,* which produces results comparable to the widely used mixture of Gaussians multimodal approach, at a much reduced computational cost and greater control of occluded object persistence.
- A novel and efficient chromatic clustering-based background model for embedded vision platforms that leverages the color uniformity of large, permanent background objects to yield significant speedups in execution time.
- A multi-scale temporal model for midground analysis which provides a means to "tune-in" to changes in the scene beyond the standard background/foreground framework, based on user-defined temporal constraints.

Multimodal mean reduces instruction complexity with the use of fixed integer arithmetic and periodic long-term adaptation that occurs once every d frames. When combined with fixed thresholding, it performs 6.2 times faster than the mixture of

Gaussians method while using 18% less storage. Furthermore, fixed thresholding compares favorably to standard deviation thresholding with a percentage difference in error less than five percent when used on scenes with stable lighting conditions and modest multimodal activity.

The chromatic clustering-based approach to optimized background modeling takes advantage of the color distributions in large permanent background objects, such as a road, building, or sidewalk, to speedup execution time. It abstracts their colors to a small color palette and suppresses their adaptation during processing. When run on a representative embedded platform it reduces storage usage by 58% and increases runtime execution by 45%.

Multiscale temporal modeling for midground analysis presents a unified approach for scene analysis that can be applied to several application domains. It extends scene analysis from the standard background/foreground framework to one that includes a temporal midground object saliency window that is defined by the user. When applied to stationary object detection, the midground model provides accurate results at low sampling frame rates (~ 1 fps) while using only 18 Mbytes of storage and 15 Mops/sec processing throughput.

CHAPTER 1 : INTRODUCTION

Research Motivation

Video surveillance has become ubiquitous to address the safety and security concerns of modern society. Monitoring systems encourage less risky behavior, provide real-time alerts to ongoing threats or potential accidents, and aid in the creation and analysis of archival data for investigative purposes. Their applications span the consumer, industrial, and military spaces. A typical large city, such as Atlanta or London, has cameras dispersed throughout public spaces, including traffic cameras to catch red-light runners or speeders and to monitor traffic conditions, subway cameras to deter crime and improve safety, security cameras in stores, airports, and other facilities to detect suspicious behaviors, watch for missing persons or suspected criminals.

As the use of video surveillance broadens, it is unrealistic to expect human monitoring of the millions of cameras. The fatigue, environmental distractions, tedium, and labor costs are obvious limitations of manual monitoring by people. These issues are further exacerbated by the high activity and volumes of data collected by monitoring systems. Whether it comes from a single camera feed of a crowded train station or multiple feeds covering different locations at an office building, a human operator cannot reliably monitor them. In large areas such as an airport, it makes sense to deploy large amounts of cameras to overlook areas monitored by human officials, as well as the inevitable blind spots.

Relying solely on humans for content intensive surveillance tasks is too tedious and error prone. Automated monitoring assistance using computer vision techniques is critical. Automated video surveillance systems can (e.g., by raising alarms when

suspicious activity is detected) assist human operators and allow them to serve as a final check of the automated results.

Traditional automated surveillance systems have taken initial steps toward limited surveillance tasks. These include traffic monitoring, face recognition, and trip-wire-based monitoring to detect when a security boundary has been crossed [21]. However, it is common for existing systems to rely on an expensive, heavily networked infrastructure: closed circuit television cameras connected to desktops and workstations, providing raw (unprocessed) video data [45].

New advances in embedded computing technology have opened up the potential for lower cost *embedded* surveillance systems that can be deployed in new environments with new surveillance capabilities. For example, Texas Instruments recently released a fixed-point multicore DSP designed to achieve high performance with low power consumption. It features six TMS320C6742 DSPs that, with all cores running at 500 MHz and 80% CPU utilization, consume 3.68W of power [47]. General purpose processors such as the ARM Cortex provide a means by which to execute control code as well as run an entire OS in an embedded system [49]. Analog Devices has produced the Blackfin processor series, which is aimed to merge the functionality of RISC and DSP architectures for use in embedded multimedia applications and video surveillance. The ADSP-BF561 embedded symmetric multiprocessor features two Blackfin processor cores and dynamic power management, and can operate at 600 MHZ [48].

The improvements in the cost and features of digital still/video camera technology in the consumer market have created popular alternatives to closed-circuit television (CCTV) capture devices that are traditionally used in security systems. CCTV systems

use analog cameras connected to capture cards for digital conversion, or digital video cameras for direct digital encoding to disk. Whereas CCTV cameras cost on the order of hundreds to thousands of dollars, consumer grade devices such as webcams, camcorders, and digital still cameras can often be obtained at one-tenth the cost and, in many cases, under 100 dollars. The USB/wireless connectivity and high resolution (2+ megapixels) of the consumer-grade devices give them advantages over traditional CCTV approaches in the embedded space. Furthermore, these technologies can be integrated into all-in-one packages such as *smart cameras*, whose purpose is not to simply capture images, but to acquire video and process its information onboard, sending only high-level recognition data directly to the user. The pervasiveness of wireless networking solutions makes smart cameras even more attractive, as they can be distributed across locations to share information and computational tasks, and reduce communication load.

Embedded smart cameras will revolutionize the monitoring of remote, outdoor locations, such as borders of countries, and the types of information and processing available to cell phone users and other camera-based handheld devices. Compact, low power real-time video analysis mobile devices can significantly enhance emerging driver assistance systems by going well beyond the currently simple cruise control and collision avoidance capabilities. More intelligent monitoring can detect erratic behavior in other drivers and can anticipate and help avoid accidents.

Problem Statement and Research Contributions

This thesis focuses on embedded, real-time algorithms for automated video analysis in surveillance applications. It addresses the problem of efficiently detecting changing or moving foreground objects in complex outdoor environments. It introduces a

novel background modeling algorithm called *Multimodal Mean* (Apewokin and Valentine et al.) [1],[4]. This thesis also optimizes the runtime performance of background modeling by exploiting temporal and spatial stability in video sequences (Valentine et al.) [42],[43]. Finally, this thesis presents a novel modeling framework for analyzing scene content across a wide range of time scales (Valentine et al.) [44]. This extends the power, applicability, and flexibility of change detection to enable new classes of applications beyond which a binary foreground/background characterization is sufficient These thesis contributions and associated challenges are described further in the next few subsections.

Contribution 1: Multimodal Mean (MMean) Background Model

High-level recognition tasks in surveillance systems require information about what is new and interesting in the scene as opposed to what is old and unchanging. Determining what is salient has been traditionally done through background modeling and elimination. Common background models analyze each pixel location across time such that changes in their overall state are detectable. Large storage requirements and high processor utilization is common in these models, which are further complicated by characteristics common in outdoor scenes (i.e. uncontrollable lighting conditions and dynamic backgrounds). Multimodal modeling techniques, which model multiple data points at a single pixel location, have been developed to address these concerns. However, the high volume of image pixels that must be processed per frame makes it difficult to deploy these algorithms in the emerging real-time, embedded computing space. In typical embedded solutions, a low-power DSP and general purpose processor with a low clock speed and limited memory resources will be used. To make computer

vision realizable in these systems, resource efficient models that balance accuracy and performance need to be developed.

In this thesis, a new background modeling algorithm is presented, called multimodal mean (MMean), which is designed to produce quality foreground images while being efficient in computational and storage requirements (Apewokin and Valentine et al.) [1],[4]. Core algorithm functionality, i.e. the ability to extract foreground data from a complex scene and adapt new background objects, is tested along with the effects of statistical and fixed thresholding on foreground image quality and computational load. Experiments show that the multimodal mean is an efficient, adaptive technique that produces results comparable to the widely used mixture of Gaussians (MoG) multimodal approach [40], at a much reduced computational cost.

Contribution 2: Optimizing Background Modeling Using Chromatic Clustering

Much of the existing work in background modeling has focused on development of algorithms with respect to theoretical accuracy. However, background modeling algorithms place a tremendous demand on the processing and memory resources of video analysis systems, straining the limited resources of embedded platforms. Each pixel must be modeled to reflect changes in the state of the scene, entailing storage requirements equal to or several factors greater than the data required to store an uncompressed image frame. This is further complicated by the use of multimodal and sliding window techniques, since they need to model multiple data points at a single pixel location. Additional computing operations are performed to maintain the state of parameters, such as means, variances, weights, and observation counts.

Often, many pixel locations in the scene are static and unchanging, which makes their maintenance operations in the models unnecessary. In a low-power embedded device, continually fetching, computing, and storing data to memory incurs significant run-time penalties that inhibit optimal run-times. Surveillance scenes typically feature areas with sustained inactivity. In many circumstances the continually inactive areas are populated with large, homogenously colored background objects, such as a wall or road. Optimization efforts that exploit this redundancy can yield significant gains in run-time performance and significantly reduce memory usage.

In this thesis a novel chromatic clustering technique is developed to exploit spatial and temporal redundancies in background objects for faster processing (Valentine et al.) [42],[43]. It is used to complement multimodal mean in a hybrid background modeling approach. In particular, the chromatic clustering technique locates large, permanent background objects for suppressed adaptation and faster processing. It is based on the idea that a significant amount of surveillance scenes contain static, permanent, homogenously colored background objects that do not need to be continually processed in an adaptive pixel-based model. We evaluate several forms of data clustering techniques to group spatial image data into regions that can be identified for faster processing. Savings in memory and computation time is measured in memory load/store operations and frames processed per second respectively. Using this hybrid optimization technique significantly reduces memory and computation operations while preserving the accuracy of multimodal mean.

Contribution 3: Multi-Scale Temporal Modeling For Midground Analysis

Many computer vision applications are tasked to identify salient objects and analyze them for suspicious behavior. Traditionally, saliency is characterized by objects that are changing or moving in short, fixed time windows. This includes moving vehicles or pedestrians crossing a busy intersection. These methods, which model data on short timescales, replicate people's ability to see rapidly moving foreground. If a scene changes at a significantly faster or slower pace, changes become more difficult to see. In applications such as illegally parked vehicle detection, abandoned object recognition, or loitering detection, these are the type of changes critical to monitoring. For example, in an illegally parked vehicle scenario, a car may enter the scene and stop in a restricted space for an indefinite period of time. In an abandoned object scenario, a person enters the scene, pauses for an unknown amount of time to place down an object, and then leaves. In a loitering scenario, persons enter the scene and linger around a location for an unusual amount of time. Table 1 provides a summary of similarly structured scenarios from the PETS and i-LIDs datasets.

Table 1: Anomalous activity scenarios

Datasets	Contents
PETS 2006	• Two persons enter the scene separately, one puts a suitcase on the ground, and both leave together. • Person enters the scene, stands with baggage, and leaves it temporarily before picking it up again. • Person enters the scene, loiters in the area and then abandons it, five people walk in close proximity to the abandoned luggage.
PETS 2007	• Control sequence in which no suspicious events occur. • Person enters the scene, loiters, and leaves. • Person enters the scene, abandons a bag, then loiters in another location, and exits the scene. • Two persons enter the scene together, one puts a bag on the ground, the other person picks it up, and both leave together.
i-LIDS 2006	• Persons enter the scene, stand by their luggage for a short time, and then abandon it. Scenes vary in crowd size and baggage distance from camera.

These scenarios typically require information concerning the object's time of entry into the scene and duration of existence. A standard background/foreground framework is inadequate to characterize this, and should be extended to a user-defined temporal dimension.

Traditionally, saliency is characterized by objects that are changing or moving. However, in a broad class of video surveillance applications, saliency is defined by an object's temporal properties. This is often application dependent, as different applications require different temporal constraints for analysis. The temporal properties that determine saliency are: when an entity appears, when it stops moving, when it starts moving, and how long it stays stationary or lingering. Analysis is used to temporally qualify clusters of pixels belonging to salient objects. In the PETS 2006 dataset, abandoned luggage detection defines saliency as stationary at 25 seconds where as loitering recognition

defines saliency as lingering at 60 seconds. Figure 1 shows an abandoned luggage sequence from the PETS 2006 dataset.

Figure 1: Abandoned luggage scenario

In this situation, abandonment is defined as the bag being unattended for at least 25 seconds. Another example is a roadside bomb detection application, where saliency is defined on a wider scale. For example, if the camera is focused on a traffic intersection, the application would continually scan for objects deposited on or near the road such that they remain for at least 150 seconds. This prevents stopping and moving vehicles within the intersection from being falsely detected. Since traditional background modeling algorithms have defined saliency as new and constantly moving foreground, a modeling approach that gives the user flexibility in tuning temporal parameters to fit the application is needed. Incorporating such information natively in background models will be advantageous because higher-level tasks can leverage this information directly, and users can tune the temporal parameters to fit their application's temporal constraints.

This thesis contribution presents a novel modeling framework for analyzing scene content across a wide range of user-defined time-scales (Valentine et al.) [44]. This is advantageous because a number of surveillance applications require analysis of content that does not fit within the constraints of regular background modeling algorithms. The

model, called *midground*, accepts as input a set of user-defined temporal constraints which characterize object ages, observation/occlusion rates, and midground extraction windows. Our evaluation tests the ability to detect stationary objects on different time-scales under moderate to heavy occlusion. This approach stands in contrast to other techniques because it offers the user wider flexibility in defining time-spans for salient object detection.

Summary of Research Contributions

In summary, this thesis presents the following three contributions:

1. An adaptive, efficient multimodal background model that enables explicit specification of temporal constraints on foreground lifetime and on background adaptation for finer control of occluded object retention.
2. An optimization technique for pixel-based background models that exploits the spatial and temporal redundancy of large, homogenously colored background objects for faster processing.
3. A midground model that allows the user to define temporal regions of saliency by tracking the temporal history of observed pixels.

Figure 2 illustrates how these contributions draw from an extensive body of related work involving background modeling, data clustering, and high-level scene analysis.

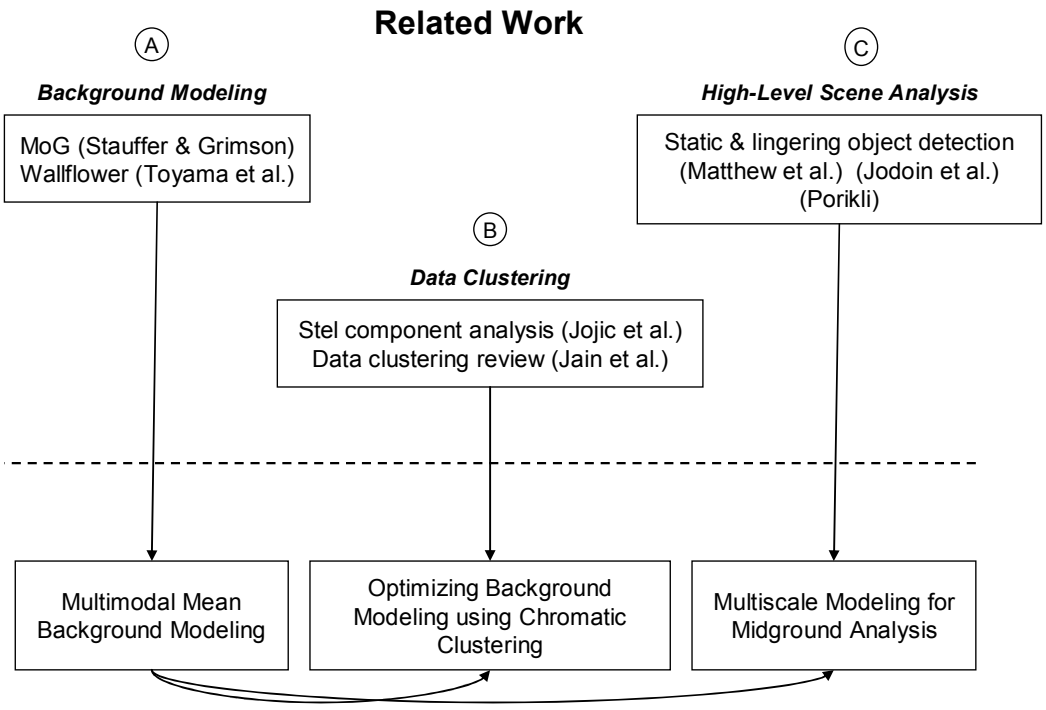

Figure 2: Thesis contributions and related work

The first contribution combines the efficiencies of unimodal techniques with the functionality of multimodal techniques to create an accurate but efficient approach for background modeling on embedded platforms. The second contribution draws inspiration from data clustering techniques to identify spatial and temporally redundant regions to further improve runtime. The third contribution presents a unified model for temporal scene analysis. Given the adjustable temporal parameters of the model, the user can set them to fit a broad set of application problems in scene activity analysis.

Summary of Results

The results of this dissertation can be summarized as follows:

- An adaptive, efficient multimodal background modeling algorithm is defined. Fundamentally, the algorithm's adaptation and pixel classification schemes are designed to reduce instruction complexity, which is critical for real-time processing on embedded platforms. A comparative analysis of thresholding schemes show that it can perform comparable to more complex methods (MoG) with fixed global thresholds in scenes that contain primarily static backgrounds and moderately active multimodal background; in particular, the average percentage difference in error between fixed and variable thresholding in these scenes is less than five percent. Furthermore, this modeling technique has the advantage of recording long-term scene history and controlling the length of occluded background persistence. This algorithm executes 6.2 times faster than the MoG with comparable results and 18% less storage resources when mapped to a 200MHz x86-based embedded PC.
- An optimization technique that improves the run-time performance of pixel-based background models based on a chromatic distribution analysis of the initial frames of the scene is developed. This technique exploits the inherent temporal and spatial redundancies of large, permanent, homogenously color background objects such as roads, buildings, and trees, so that their adaptation is suppressed and processing resources is spent on analyzing active, salient data. When applied to the multimodal mean, and run on an embedded x86 200 MHz processor, a 45% improvement in performance and a 58% reduction in storage operations are achieved.

- A midground model that gives the user the ability to highlight object saliency based on a user-defined temporal window is presented. It requires only 18Mbytes of storage and 15Mops/sec processing throughput. It performs accurately in real-world sequences at relatively low frame-rates (~1 fps).

Overview of Content

This thesis is organized as follows. Chapter 2 introduces a fast, background modeling technique with support for embedded platforms. Chapter 3 introduces an optimization technique to improve the run-time performance of pixel-based background models by exploiting the temporal and spatial redundancies of large permanent background objects. Chapter 4 introduces the concept of midground detection, examining its effectiveness when applied to stationary object detection. Conclusions and future work are presented in Chapter 5.

CHAPTER 2 : MULTIMODAL MEAN BACKGROUND MODEL

Introduction

Background modeling algorithms typically serve as the first step in a high-level vision system for tasks such as pedestrian tracking, moving target identification, or vehicle monitoring [11], (Apewokin and Valentine et al.) [2]. The objective of background modeling is to identify salient content in the scene by "subtracting" out portions that are static or have uninteresting motion, such as waving tree branches. The quality of technique is important because high-level recognition tasks require accurate segmentation of foreground and background to be effective. However, the growing embedded computing space makes it a challenge to map algorithms to these devices for real-time execution while maintaining accuracy.

Determining how to set parameters such as match thresholds and adaptation rates in background models is not a straightforward process. Thresholds are frequently presented as arbitrarily determined global values, or as functions of statistical modeling of scene noise and observed foreground activity. The former is rigid and lacks detailed insight into the environments where global thresholds are suited, and the latter can be too complex and computationally expensive for common scenarios. Parameters for adaptation are often presented without relation to the expected lifetime of foreground: that is, how long should a newly stationary object be considered foreground, and how long should occluded background be retained in the model? Generally, threshold selection influences the number of error pixels in the foreground frame while the adaptation rate affects how long recent objects in the scene are considered foreground. Additional difficulties are encountered because some model parameters are

interdependent. For example, a change in one parameter to address a problem can have undesired effects on other parameters.

To address these issues, we introduce an efficient, pixel-level, adaptive multimodal background model, called multimodal mean, which uses temporally annotated pixels to distinguish between background and foreground objects (Apewokin and Valentine et al.) [1],[4]. Fundamentally, the algorithm's adaptation and pixel classification schemes are designed to reduce instruction complexity, which is critical for real-time processing on embedded platforms. This is accomplished primarily through the use of fixed-point operations and periodic adaptation that occurs only every *d* frames. To achieve high accuracy, it incorporates the multimodal characteristics of the well known MoG background model. The significant reduction in instruction complexity of the multimodal mean over MoG allows it to run several times faster on embedded platforms (Apewokin and Valentine et al.) [1],[3],[4],[5]. Furthermore, we show that the average percentage difference in error between fixed and variable thresholding in the environmentally stable scenes (static lighting and modest dynamic backgrounds) tested, is less than five percent. Therefore the use of fixed thresholding, where appropriate, further reduces instruction complexity. In contrast to other techniques, such as the MoG background model, multimodal mean provides an efficient adaptation scheme to control the persistence of occluded background, preventing premature mode deletion due to occlusions.

Related Work

A variety of pixel-level background modeling techniques exist, varying in computational complexity, adaptation, thresholding, and accuracy. Non-recursive

techniques maintain sliding window buffers of recent image frames. At each successive input, the oldest frame in the buffer is replaced. Image pixels are classified as foreground if they are outside a specified range of the mean or median of the buffer. Frame differencing, temporal median filtering, sliding window mean, non-parametric modeling, and linear predictive filtering are representative non-recursive background models. These techniques work well in scenes with rapidly moving foreground and frequently visible background. However, they lack the ability to model long-term scene changes, dynamic and occluded backgrounds, and have difficulties with slow moving objects [10].

Frame differencing, temporal median filtering, and sliding window mean are computationally inexpensive, which is beneficial for low-cost, low-power embedded implementations. Although more complex, the non-parametric model by Elgammal et al. has the advantage of handling multimodal backgrounds since it uses a full density function to estimate the likelihood of a pixel matching the background [17]. Also the linear predictive filtering approach has the advantage of smaller deviations from predicted background values, which is helpful in preventing foreground objects that are similarly colored to background objects from being misclassified.

Recursive modeling techniques do not use sliding windows. Examples of these include Kalman-filter based approaches, approximated median filters, and the widely used Mixture of Gaussians (MoG) [10],[31],[40]. Surveys of these can be found in [10],[33],[35]. The approximated median filter has the advantage of a low-memory footprint and computationally inexpensive update operations using addition or subtraction. The Kalman-filter based approaches are robust to noise. MoG models pixels as Gaussian distributions, with multiple modes captured using multiple Gaussians per

pixel location to model dynamic background scene elements, such as rippling waves [40]. Other approaches include foreground segmentation based on varying frame rate, edge detection, region analysis in homogenously colored areas, or optical flow [20],[14],[15],[24],[28],[33].

Background modeling techniques distinguish incoming pixels as foreground or background based on previously observed information in the scene, using some measure of pixel dissimilarity between previously viewed and new pixels. This can be a difficult task, as incoming pixels can differ from the known background because they are new, a result of an illumination change, or a color fluctuation due to noise. To avoid false pixel classifications, difference thresholds, ε, used to compare pixels need to be set wide enough such that noise does not interfere, and narrow enough so that new data are not missed. In a number of background modeling techniques, these thresholds are determined experimentally [1]. For example, the background model presented in [6], while similar in structure to MoG, used a running average to model pixel clusters instead of a Gaussian distribution, and a fixed threshold of 15 instead of variance for cluster matching. The rationale behind the choice of 15 was not stated. Alternatively, standard deviation-based thresholds are popular in many algorithms because of their adaptability to changing environmental conditions such as lighting and weather. In [9][40] and [46], a recursive estimate of standard deviation is used to determine if an incoming pixel matches a background pixel. An initial guess of the standard deviation is declared σ_o, with reliance on subsequent adaptations to converge the deviation near its true value. A match is declared if the pixel falls within $k\sigma$ absolute standard deviations of the background pixel, where k has been set equal to 2.5. The work in [8] highlights some basic principles for the

selection of σ_o, stating that σ_o should be less than the range used if one found a fixed threshold ε suitable for the scene. Specifically, σ_o would be equal to $\frac{\varepsilon}{2.5}$. Although more adaptable, the standard deviation-based threshold is ultimately still dependent on empirically chosen parameters (k, σ_o, and possibly ε), with the rationale behind them explicitly discussed. Some insight can be gained from the work in [1], which observes that the dispersion of a background pixel at any location should be no more than three times that of its standard deviation. Fixed thresholds ε are unlikely to outperform the quality of adaptable thresholds, such as those based on standard deviation, because they are more affected by illumination changes and scene noise [35].

Methods for adapting background models to evolving scene conditions vary substantially. A simple technique such as temporal averaging uses the k-most recent frames to model the background, automatically discarding scene history older than $k+1$ frames. Frame differencing represents an extreme of this case, where a foreground object from the previous frame will become background when compared to the current frame, with no other information about the scene included. Sophisticated methods such as the MoG are designed to provide more control over adaptation, using a combination of learning rates and pixel weights to control the adoption speed of new objects. In the MoG algorithm pixel-modes are sorted by the ratio of their weight and variance in ascending order. To determined which are background pixels, the sorted modes are summed, one-by-one, as long as the sum is less than or equal to T. The modes included in this sum are considered background. The learning rate α controls the speed at which a new pixel's weight will gain enough statistical evidence to become a dominant background mode. In MoG the value of T has been presented as an arbitrary choice, with smaller values

favoring unimodal backgrounds and larger values favoring multimodal backgrounds. The appropriate value of T for a given type of scene is not clear and the explanation of the purpose behind α is imprecise. Also, the replacement policy unavoidably forces the least probable mode to be replaced when a new pixel is observed, which causes relearning of occluded background modes.

When evaluating output quality of the foreground extraction frames, existing work has focused on the accuracy of detecting a foreground object's initial entry into the scene [41], with additional attempts made to avoid issues such as ghosting. The work in [9] derives that the time it would take for a new pixel to be declared background in the MoG algorithm is $\frac{\log(T)}{\log(1-\alpha)}$ frames. If background ghosting is something of concern, then the user would need to set the learning rate low enough to account for the expected lifetime of temporarily static foreground objects. Unfortunately, the user would still have little control over how long the occluded background is retained in the background after the new object becomes dominant. The occluded background could be pushed out of the model in favor of any transient foreground, causing a potentially unwanted foreground ghost when the background is relearned. Furthermore, the length of time that a new object is considered foreground is dependent on both T and α. Work in [9] attempts to avoid this type of problem by measuring the number of observations of pixel modes as well as the recency of observance to distinguish background and temporarily static objects. Another approach is to analyze surrounding background regions of the ghost to determine if it is true foreground. This approach was taken in [39], where they wanted to distinguish between an abandoned or removed object. Another approach, taken in [13], uses optical flow information about foreground blobs to distinguish true foreground and

un-occluded background. In our approach, we give the user an efficient method for control over the appearance of background ghosts by supplying a set of parameters to define foreground lifetime and the duration for persistence of old, occluded background.

Multimodal Mean Algorithm

In this research, we introduce the multimodal mean background model, which records temporal information about when new pixel values are observed and how often they have been seen. Its model parameters allow explicit specification of temporal constraints on foreground lifetime and occluded background persistence. Its algorithm design is intended to map efficiently to embedded platforms with limited memory and computational resources.

The model maintains a set of *k* cells, each containing the three color component (RGB) sums of an observed pixel value and the number of times it has been seen. A set may contain several cells, belonging to different color values observed at the pixel coordinate. The structure of a cell is shown in Figure 3.

$S_{i,t,r}$	$S_{i,t,g}$	$S_{i,t,b}$	$C_{i,t}$

Figure 3: Cell structure

Where $S_{i,t,j}$ is the sum of observed pixel values for pixel location *i* at time *t* for color component *j*. *C* is the observation count.

The multimodal mean uses the mean of previously observed pixels to determine whether or not pixels in the current frame are new or old. A pixel in the current frame matches an existing cell if Equation 1 is satisfied.

$$\left| I_{t,j} - \mu_{i,t-1,j} \right| \leq \varepsilon_j, \forall j \tag{1}$$

Where $I_{t,j}$ is the current pixel value for color component j at coordinate i, $\mu_{i,t,j}$ is color component average for the cell, and ε is the match threshold. $\mu_{i,t,j}$ is computed by dividing the color component sums by their observed $C_{i,t}$ (Apewokin and Valentine et al.) [1],[4]. A pixel is declared *foreground* if the amount of time it has been observed is less than the time specified by the foreground threshold *Fth*.

If a pixel is new to the scene, its color values are recorded in a cell, with, its count field initialized to one. To determine if this new pixel will replace an existing cell or comprise a new entry in the cell list, the observation count of the last created cell is checked. If its observation count is less than the cell threshold *Cth*, then it is deleted from the model and replaced with information about the new pixel. If its observation count is greater than the cell threshold, then all cells are kept and a new entry in the cell list is created for the new pixel. In contrast to the foreground threshold *Fth*, which is used to distinguish between background and foreground pixels, the cell threshold *Cth* is used to facilitate the replacement of short-lived moving foreground objects that quickly enter and exit the scene.

Defining Foreground and Background

The multimodal mean model determines whether or not a pixel is background or foreground based on whether it matches a cell and if so, its observation count. In general terms, background is considered to consist of long-term stable objects and foreground consists of new, short-lived ephemeral objects. However, in many applications, the time at which a foreground object should be transitioned to a background classification is not absolute. A specific example involves a person walking into camera view and lingering. A loitering detection application would track the lifetime of the person in the scene,

viewing the person as foreground only until it can determine whether or not they are a threat. In contrast, a pedestrian counting application is just concerned with counting the number of people that enter the scene. It only needs to consider the person as foreground during its initial viewing. Therefore, the definition of foreground lifetime, and the migration of objects from foreground to background should be specified by user-defined constraints, tailored to a given application. In our multimodal mean algorithm, foreground lifetime is specified as a temporal parameter, *Fth*. This temporal foreground threshold is specified by the user in units of time; any pixel value having an observation count less than *Fth* is treated as foreground.

A small foreground threshold absorbs new objects quickly into the background whereas a very large foreground threshold resists adaptation. This is partially complicated by the characteristics of multimodal backgrounds and the speed/size of moving foreground. When considering the hypothetical case of a dynamic background with three distinct colors, each equally occupying a space in time, it will take three times the value of the foreground threshold before any of the three colors is declared background. Another case is a long train that is moving slowly. If the train has a uniform color, its tail-end could be misinterpreted as background because the cells that collected data from the front of the train will have accumulated a large enough count to surpass the foreground threshold by the time it sees the back of the train. Most foreground objects of interest (people, cars, etc.) have a smaller horizontal width than a train, and for many scenes single-mode backgrounds outnumber the multimodal. Setting the foreground threshold based on a defined temporal lifetime is sufficient for these cases.

Scene Adaptation through Decimation

When monitoring scenes for extended periods of time, the background model must be able to adapt to changing conditions. This is important because, over time, cells will be created due to lighting transitions, newly visible background objects, or short-term foreground objects. Old cells that have been made irrelevant by changes in scene conditions need to be removed. This includes short-term foreground objects having small observation counts, old cells that have been occluded for an extended period of time, or cells that have changed color due to a persistent lighting change (day-to-night). We have devised a technique for the multimodal mean called *decimation*, which serves as the primary measure of long-term adaptation in the scene. In decimation, cells are periodically deleted from the model according to a predetermined decimation rate *d*.

Decimation works as follows; every *d* units of time, all cells are scanned in each set. The observation count *C* and component sums *S* of all cells are halved. If a cell's count value is less than the cell threshold *Cth*, then it is removed from the set. Halving the value of cells makes it possible for dominant cells, those with the largest observation counts, to be overtaken by new cells after they become occluded. The decimation rate is set based on how long the user needs an occluded background to maintain dominance. If the decimation rate is too frequent, then background cells will be deleted during brief periods of foreground occlusion. At a minimum, the decimation rate should be at least twice the value of the foreground threshold *Fth* to ensure that unwanted ghosting effects are not observed. When a background cell is persistently occluded, it would take approximately $\log\left(\frac{2d}{Cth}\right)$ decimations before it would be deleted from the model. For a

cell threshold of four, Table 2 shows the approximate time to deletion of a persistently occluded background cell as a function of decimation rate.

Table 2: Occluded background approximate time to deletion (Cth = 4 seconds)

Decimation Rate (secs)	***20***	***30***	***40***	***50***	***60***	***70***	***80***	***90***	***100***	***200***
Appx. Time to Deletion (secs)	60	117	173	232	294	359	426	494	564	1,329

In some cases, a new cell will be created immediately prior to a decimation run. Since the new cell will not have enough time to gain significance (high observation count), it will be deleted during the decimation and re-learned. Any new cells that were created within a $2\times Cth$ window before decimation will be affected by this. If Fth is less than $2\times Cth$, then it is possible for the cell to absorb into the background then reappear as foreground after decimation. If Fth is greater than $2\times Cth$, then the cell will appear as foreground longer than the time set by Fth, taking at most an extra $2\times Cth$ counts to absorb into the background.

Evaluation and Results

We perform experiments that evaluate the foreground detection quality of the multimodal mean under different background matching threshold conditions, the accuracy of decimation for long-term adaptation, and the runtime efficiency of multimodal mean. The purpose of these experiments is to determine how effective fixed thresholding is for multimodal mean, how accurate decimation is for controlling occluded object persistent (long-term adaptation), and how fast can the multimodal mean run on a resource-constrained embedded platform in comparison to other techniques.

To determine how effective fixed thresholding is in comparison to standard deviation, we compare the percentage difference in false positive/negative foreground

pixel errors between the two in the form of the Jaccard coefficient [38]. This experiment begins with an analysis of the variability in background pixels, followed by an evaluation of foreground detection quality as a function of background matching threshold. The false positive/negative foreground segmentation errors on randomly selected image frames, in the form of the Jaccard coefficient, is calculated. The representation of the Jaccard coefficient is shown in Equation 2, where *TP* is the total number of true positives, *FP* is the number of false positives, and *FN* is the number of false negatives.

$$J_c = \frac{TP}{(TP + FP + FN)} \tag{2}$$

Representing the foreground detection errors in this fashion mitigates the effects of scenes with overwhelmingly large amounts of true negative pixels on reported accuracy [36],[38].

Decimation, which gives the user control over how long background objects are remembered by the model once they are occluded, is evaluated by running test sequences at different decimation rates and comparing the theoretically estimated occluded lifetime, as illustrated in Table 2, with the experimentally observed lifetime. Finally, to determine how efficient the algorithm is in comparison to others, the multimodal mean and several popular background models are mapped to an embedded PC to perform a runtime comparison. The algorithms are coded in C and compiled for the eBox 2300 200MHz x86 SoC and benchmarked.

Model Parameters

The multimodal mean model is controlled with four parameters, the foreground threshold *Fth*, cell threshold *Cth*, background matching threshold ε, and the decimation

rate *d*. The foreground threshold *Fth* controls the length of time that will pass before a foreground object is absorbed into the background. The cell threshold *Cth* is used to reduce the accumulation of cells in sets from short-term foreground transients. The image sequences used in the experiments primarily feature steadily moving pedestrians at walking paces and cars moving at speeds between 15 mph and 30 mph. Their lifetimes within the scene are relatively short (less than 10 seconds). To detect foreground that enters a scene briefly and leaves, we set our foreground threshold *Fth* to be 10 seconds. We set the cell threshold *Cth* to four seconds based on the assumption that a person walking orthogonal to the camera view at a constant pace will occupy a pixel location no longer than one second. Setting the threshold at four seconds makes an allowance for objects moving at non-orthogonal directions to the camera, and objects that are larger in horizontal width (i.e. bus, car, etc.). The background matching threshold is used to compare the current image pixel to a known background color.

Measuring Pixel Dispersion

In background models, current image pixels are compared with known background to determine if they are new or previously seen. The comparison is made based on observed similarity between pixel colors, rooted in some measure of perceptual color difference or statistically observed variability of background colors (standard deviation). For fixed threshold background models, the user needs to manually set thresholds that minimize the occurrence of false positives and false negatives. Also, models based on recursively updated standard deviation need to have an appropriate value for the initial deviation σ_o. At a minimum, the threshold value needs to be set higher than the variability in background pixels caused by random scene noise and

camera sampling error. To empirically determine this, the following set of experiments examines the dispersion of color in background pixels more closely.

Background pixels generally have a small standard deviation σ, leaving a narrow margin of error to detect foreground in the presence of background noise. Figure 4 contains a plot of the green component of a single background pixel, its mean, and standard deviation.

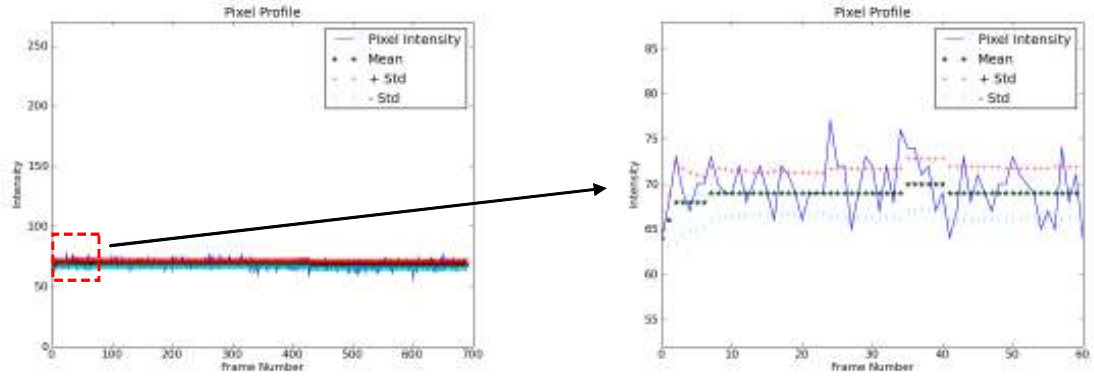

Figure 4: Standard deviation of background pixel

The pixel is part of the blue wall in the scene shown in Figure 5.

Figure 5: Staircase sequence

The standard deviation of the background pixel's color component is low ($\sigma < 10$) however, there exist several points in time where the color of the background pixel falls outside its standard deviation. Since the deviation is very small, using a multiplicative proportion of it (i.e. $k\sigma$) as the sole threshold for comparison of new and old pixels produces excessive false positives in the form of random noise speckles. Therefore, any

threshold value, fixed or variable, should be bounded to a minimum value based on an "average" measure of observed background color dispersion [8].

To determine a minimum allowable threshold value for foreground detection we conducted an experiment using three test sequences, shown in Figure 6, with the match threshold set as $\varepsilon = k\sigma$. The scenes were taken at the Georgia Institute of Technology for the purpose of pedestrian tracking.

Figure 6: Test sequences with highlighted background sub-blocks

σ is the standard deviation of all observed pixels at coordinate x. Each sequence has steady pedestrian traffic walking orthogonal to the camera. For this experiment we have computed the standard deviation explicitly, as shown in Equation 3

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{t=1}^{N}(x_t - \bar{x})^2} \qquad (3)$$

where j is the color component, x_t is the current pixel value at time t and $\bar{x}$ is the mean. Pixels within the absolute range $k\sigma$ of a cell, having a count value greater than Fth are considered background. Varying k from one to five, we found that k needs to be greater than or equal to three to ensure that most of the background noise is not misinterpreted as foreground. To get a closer view of how the background pixels vary in a scene, Table 3 lists the average standard deviation of the highlighted background sub-blocks in Figure 6.

Table 3: Standard deviation of background sub-blocks

Sequence	Standard Deviation
Parking	3.59
Walkway	1.32
Staircase	3.60

Taking the average standard deviation of these sub-blocks and multiplying by $k = 3$ resulted in a deviation range of 3.96 to 10.80. In subsequent experiments we make any ε, whether fixed or computed through standard deviation, no smaller than 10. This coincides with previous work in [8], which had a low variance threshold of 49.0.

Threshold Accuracy

Statistically-based background matching thresholds are based on the observed activity within the scene whereas fixed thresholds assume a set of expected behaviors and background characteristics (i.e. controlled lighting, etc.) about the scene. In this section we investigate statistical and fixed thresholding for background matching in the multimodal mean model by comparing false positive/negative foreground segmentation errors on randomly selected image frames. Figure 7 shows the result of foreground detection using standard deviation and a fixed global threshold. For standard deviation, k holds the values *[1, 2, 3, 4, and 5]* where $\varepsilon = k\sigma_{x,j}$ and $\sigma_{x,j}$ is the standard deviation of all pixels observed at coordinate x for color component j. For fixed thresholds, ε holds the values *[10, 20, 30, 40, and 50]*.

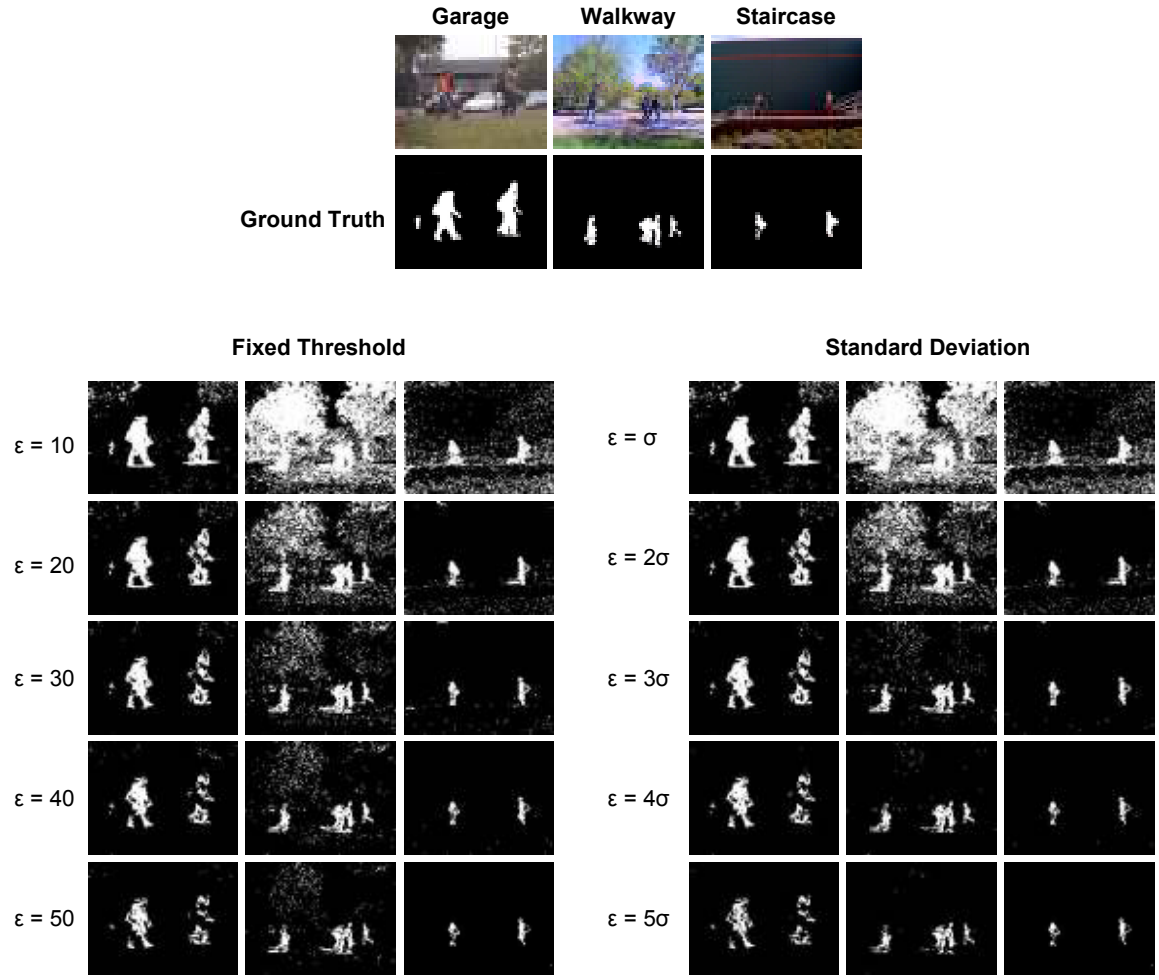

Figure 7: Foreground detection comparison using fixed and variable thresholds where ε = kσ for the variable threshold

Table 4 and Table 5 show the false positive/negative pixel errors accumulated for each frame, in the form of the Jaccard coefficient, when compared against the ground truth.

Table 4: Jaccard coefficients for MCD threshold

	Threshold (*ε*)				
	10	***20***	***30***	***40***	***50***
Parking	0.67	0.73	0.71	0.68	0.65
Walkway	0.07	0.18	0.34	0.48	0.56
Staircase	0.12	0.46	0.62	0.70	0.70

Table 5: Jaccard coefficients for standard deviation threshold

	Threshold ($\varepsilon = k\sigma$)				
	σ	2σ	3σ	4σ	5σ
Parking	0.67	0.74	0.72	0.68	0.63
Walkway	0.07	0.18	0.57	0.69	0.69
Staircase	0.12	0.47	0.73	0.70	0.68

The narrow match thresholds ($\varepsilon < 30$) operate under the assumption that the variation of background colors is low. However, if it is too narrow the foreground detection will have a high number of false positives from noise and fluctuating environmental conditions. This is visible in all the test sequence results from Figure 7 when $k = 1$ and $\varepsilon = 10$. The Walkway and Staircase sequences have a low Jaccard coefficient (< 0.50) in the narrow threshold range. Wider thresholds allow a larger variation in background color. However, if the threshold is too wide it can result in increased false negatives from ambiguities between foreground objects similar in color to background. This is visible in the higher threshold tiers of the *Parking* sequence where $k = 5$ and $\varepsilon = 50$, with the Jaccard coefficient either falling or tapering as the threshold is widened. Generally, we have found that the highest Jaccard coefficients correspond closely with a subjective visual assessment of the best foreground image. In the *Parking* sequence, the highest Jaccard coefficient came from a fixed threshold of 20 and a standard deviation threshold of 2σ. The highest Jaccard coefficient from the *Walkway* sequence came from a fixed threshold of 50, and a standard deviation threshold of 4σ and 5σ. The Staircase sequence performed best at a fixed threshold of 40 and 50, and a standard deviation threshold of 3σ.

In most of the images, the error level and segmentation quality using standard deviation and fixed thresholds are comparable. The average percentage difference in error for the *Parking* sequence is 1.2%. For the *Staircase* sequence the difference is 4.3%. The

major exception occurs in the *Walkway* sequence, where the trees oscillated very rapidly. This is because the dynamic backgrounds have a much higher standard deviation than the static backgrounds. In this case the average percentage difference in error is 21.5%, which is significantly larger than the differences in the *Staircase* and *Parking* sequences. Detailed examinations of specific image sub-blocks from Figure 8 helps to explain these differences.

Figure 8: Image sub-block analysis

Block A contains static background objects while Block B contains multimodal backgrounds such as a waving tree branch, or rippling leaves. The standard deviation is recorded for each pixel in the sub-blocks and averaged in Table 6.

Table 6: Sub-block standard deviations measurements for test sequences

	Single Mode Block (A)	Multi Mode Block (B)
Parking	3.59	18.40
Walkway	1.32	23.04
Staircase	3.60	10.84

The amount by which the standard deviation of the multimodal sub-blocks is higher than the single-mode is dependent on the physical behavior of the dynamic background and its occluding colors. The tree regions in sequence *Walkway* and *Parking* varied frequently

between green leaves and sky, causing significantly more false positives. In contrast, the leaves in the staircase sequences vary between a more similar set of colors. Using standard deviation as the threshold alleviates more false positives because it computes a wider background threshold for the dynamic background pixels. The *Parking* and *Staircase* sequences show similar results between the fixed thresholds because the dynamic background population is sparse (*Parking*) and the variation is small (*Staircase*). If one looks closely at the foreground segmentation of the *Staircase* sequence, one can see that the standard deviation threshold had less false positives in the ground area where there are fluttering leaves. However, since the variations were small, the fixed threshold produced a comparable result, with slightly more error.

The higher accuracy of variable thresholding in our test results comes at the expense of increased computational complexity and storage requirements. Table 7 shows the number of operations per pixel required to compute a set of thresholds.

Table 7: Operations per pixel for cell match thresholds

Method	Total	Add	Sub	Mult	Div	Root	Abs	Comparison
SAD	*12*	2	3	0	3	0	3	1
MCD	*12*	0	3	0	3	0	3	3
Standard Deviation	*44*	7	7	12	9	3	3	3
Rec. Est. Std. Deviation	*33*	3	6	12	3	3	3	3

This table includes the maximum component difference (*MCD*) for fixed thresholding and the standard deviation approach, used in the results from Figure 7. The table also includes two additional methods. The first is a fixed threshold using the sum of absolute difference (*SAD*) and the second is a variable threshold using the recursively estimated standard deviation (Rec. Est. Std. Deviation) shown in Equation 4.

$$\sigma^2_t = (1 - \rho)\sigma^2_{t-1} + \rho(I_t - \mu_t)^2 \quad (4)$$

The standard deviations were computed separately for each color component. Using *SAD* allows for more variation in individual color components when matching. However, ambiguities are introduced when modeling objects that retain the majority of their color information in just one of the three components. The recursive estimation provides a more cost-effective way of computing the standard deviation, with slightly increased error. Table 8 and Table 9 compare the foreground detection accuracy of these techniques.

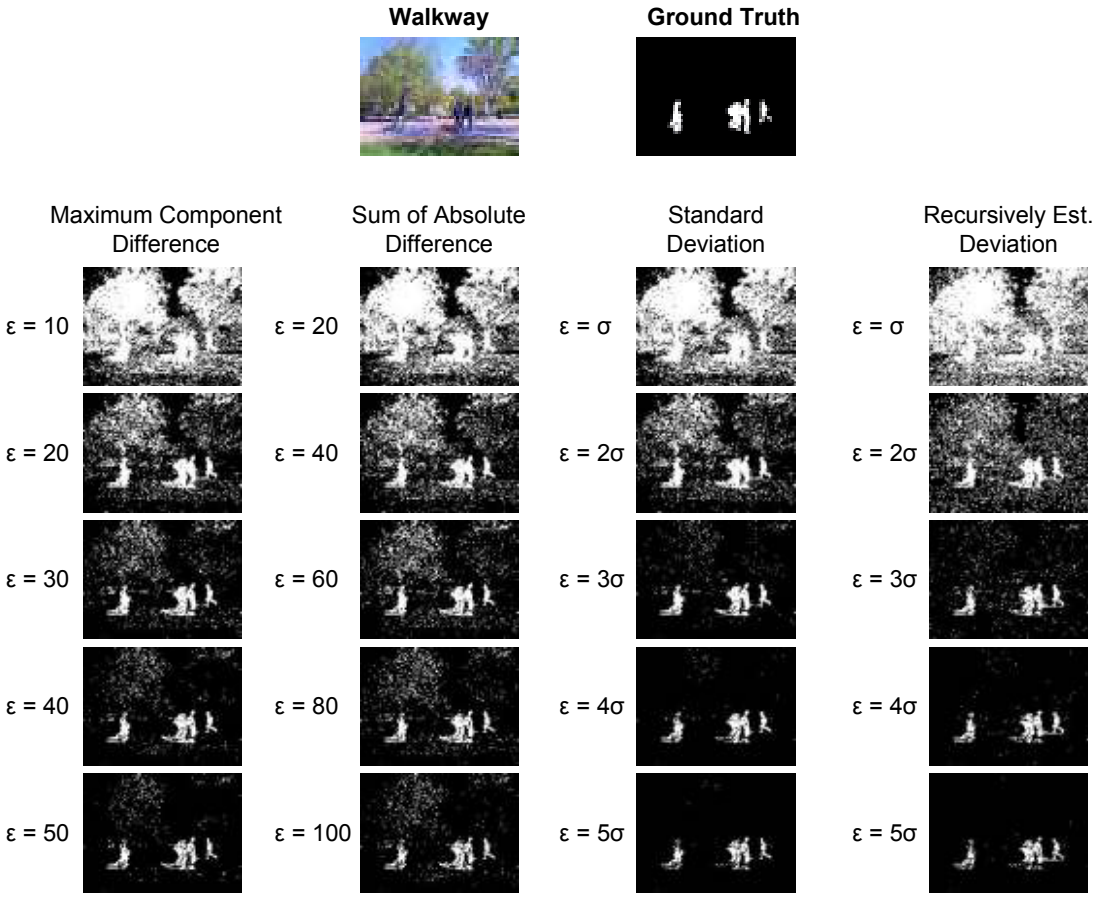

Figure 9: Comparison of thresholds on the *Walkway* sequence

Table 8: Jaccard coefficient comparison of *SAD* (x = 2) and *MCD* (x = 1) thresholds

	Threshold (ε)				
	10x	***20x***	***30x***	***40x***	***50x***
MCD	0.07	0.18	0.34	0.48	0.56
SAD	0.07	0.17	0.30	0.42	0.54

Table 9: Jaccard coefficient comparison of full standard deviation and recursively estimated standard deviation thresholds

	Threshold ($\varepsilon = k\sigma$)				
	σ	2σ	3σ	4σ	5σ
Full SD	0.07	0.18	0.57	0.69	0.69
Est SD	0.06	0.15	0.51	0.66	0.65

The learning rate for the estimated standard deviation is 0.01 with an initial standard deviation of 10. We chose an initial standard deviation of 10 to coincide with the threshold lower bound defined in Section 4.1. In computational terms, using standard deviation for thresholding requires more than twice the number of operations per pixel when compared to the fixed thresholds. Also, standard deviation-based methods use more memory, requiring three extra integers per cell for squared component sums (true standard deviation), and three extra floats to store previously estimated standard deviations. In the *Parking* and *Staircase* sequences, the differences between the fixed and variable thresholds are negligible, making the added computational complexity unnecessary. In the *Walkway* sequence, the average percentage difference in error between the estimated standard deviation and true standard deviation is 11%, suggesting that using the estimated standard deviation is an acceptable compromise that can reduce the number of operations per pixel if more sophisticated thresholding is required. The differences between *SAD* and *MCD* for fixed thresholding are difficult to perceive in the test sequences. The average error percentage difference is 7% in the *Walkway* sequence.

For our experiments, we find it preferable to use *MCD* for fixed thresholding as it requires all three color components to be close in value to observed background pixels when matching.

Generally $k = 3$ and $\varepsilon = 30$ produced acceptable foreground images in Figure 7, representing an acceptable assumption of the threshold needed to obtain quality foreground images on a variety of sequences, both known and unknown. Although not optimal, when viewing the results from Table 4 and Table 5, it represents a good tradeoff between the extremes of the narrow and wide thresholds. To understand how a fixed threshold of $\varepsilon = 30$ was comparable to the standard deviation based thresholds in the *Parking* and *Staircase* scenes we refer to Table 6, which measured the average standard deviation of background objects in areas with no foreground activity. The standard deviations of background objects vary between zero to four for the single-mode backgrounds and 10 to 23 for the multimodal backgrounds. Most of the background block deviations are below 20, except for the multimodal block in *Walkway*, where there is a rapidly waving tree branch. Having a threshold of $\varepsilon = 30$ favors scenes in which background is more than likely to be occluded by very dissimilar foreground object colors, resulting in low false positives at the expense of a potentially increased amount of false negatives and false positives in multimodal areas.

Adaptation through Decimation

Decimation is used to control how long background cells are kept in the model after occlusion, and as a method to purge short-lived cells from the model. During decimation, cell observation counts are halved periodically so that the new, persistent objects can gain dominance over the background areas they occlude. This is useful in

controlling the appearances of background ghosts, as well as preventing the overflow of storage elements.

Figure 10 contains a scenario that we have used to test the effectiveness of decimation for long-term scene adaptation.

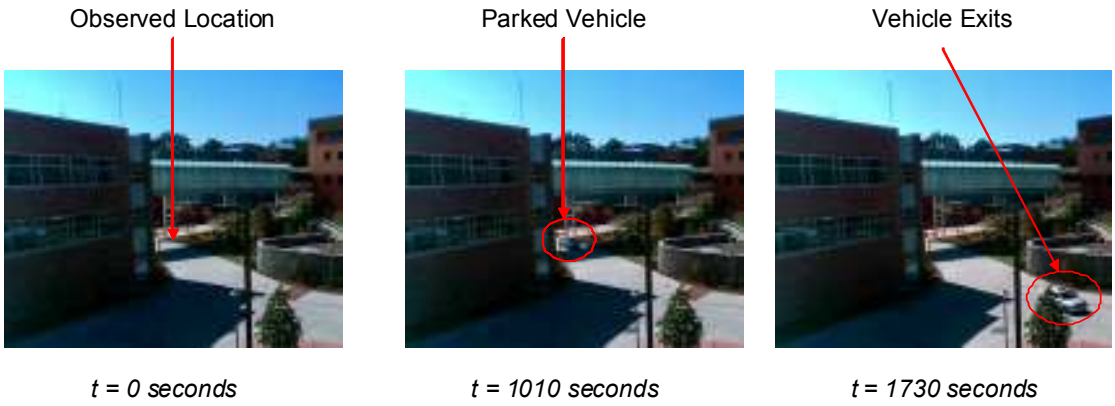

Figure 10: Long-term adaption scenario using decimation

In this scenario a car parks for 11 minutes before exiting. Prior to the entry of the car, the background model has been monitoring the scene for 16 minutes. The cells associated with the road have built up several minutes' worth of history in the form of ratiometric color sums and observation counts. Figure 11 shows a plot of the observation count for all cells at the location marked in the image by the red circle in Figure 10.

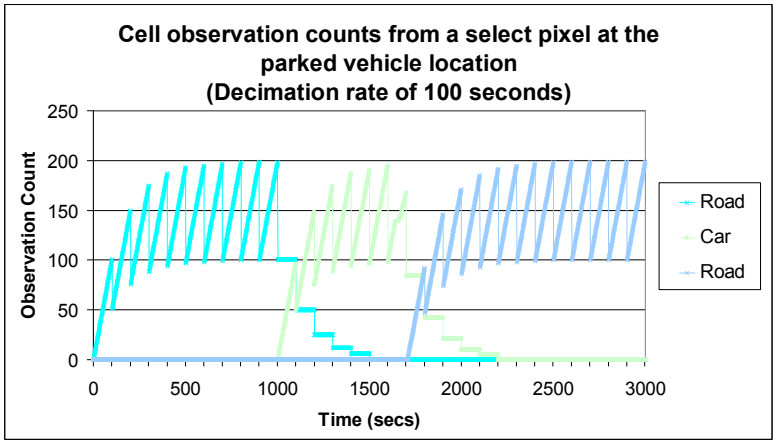

Figure 11: Cell count histories (decimation example)

In this plot the decimation rate is set at 100 seconds. The high observation count at t = 500s to t = 1000s indicates that the object the pixel belongs to (the road) is persistent in the scene. Although the cell associated with the road pixel was decimated 10 times, its continued observance in the scene for 16 minutes allowed its observation count value to maintain prominence. Once the road was occluded, decimation of the road cell allowed the car to overtake its observation count. Continued decimation eventually leads to the road cell being deleted from the model.

The maximum count any cell can reach with decimation is two times the decimation rate ($2d$). If the road was observed with no illumination changes or occluding foreground for an extended period of time and became persistently occluded, it would take approximately $\log\left(\frac{2d}{Cth}\right)$ decimations before it would be deleted from the model. This coincides with the results in Figure 11, where it takes four decimations for the cell associated with the road to be deleted. With decimation set to occur every 100 seconds, the road has to be occluded for approximately 7.2 minutes before it is deleted. Adjusting the decimation rate d to 25 seconds would cause the road to be deleted after only one minute of occlusion.

The value of d is selected based on if the user prefers an occluded background to be deleted quickly or slowly. Table 2 shows the approximate time to deletion of occluded background as a function of decimation, where the time to deletion is $T = d\log\left(\frac{2d}{Cth}\right)$. Figure 12 is a real-world example of how the road's observation count decreases, depending on the decimation rate.

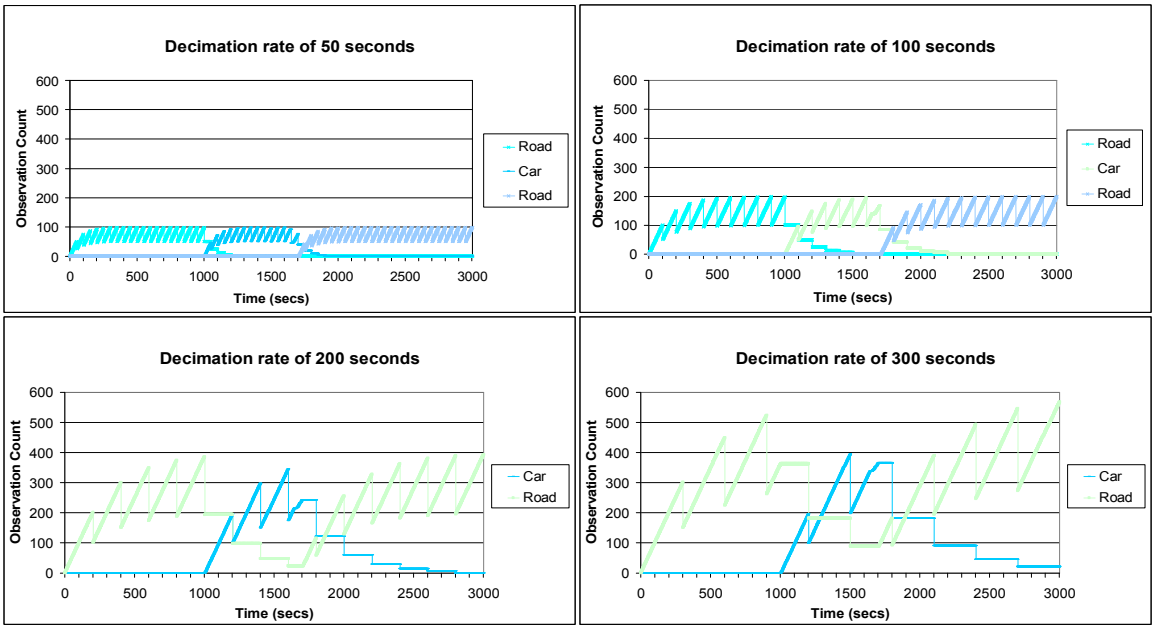

Figure 12: Observation counts from a select pixel at the parked vehicle location as a function of decimation

At a decimation rate of 50 seconds the road cell is deleted at t = 1,200s, approximately 200 seconds after it was occluded. At a decimation rate of 100 seconds, the road cell is deleted at t = 1,500s, approximately 500 seconds after it was occluded. At a decimation rate of 200 seconds, the road would need to be occluded for approximately 1,329 seconds before it is deleted from the model. Since the car occludes the road for 660 seconds, the road cell was not deleted. It is recalled once the road becomes visible. This tracks closely with the theoretical approximations of time to deletion, as shown in Table 2.

In addition to controlling occluded object persistence, decimation prevents overflows in observation counts and accumulation of old cells in sets. Arithmetic overflow is less of a concern when dealing with 32-bit integers due to its dynamic range. However, a 16-bit implementations could encounter arithmetic overflow in a matter of hours or days, depending on the frame rate and decimation rate. Therefore, the

decimation rate must be less than *DR/(2FR)*, where *FR* is the frame rate of the sequence and *DR* is the dynamic range of the observation count. Another concern is the amount of memory used to store cells in a set. To have a better understanding of the effect of decimation on memory usage, we conducted an experiment using three long duration scenes. *Docks* served as a control sequence because there is sparse foreground activity and a relatively stable background. The *Science* sequence has a high volume of vehicle traffic and moving pedestrians. The *Traffic* sequence overlooks a road with frequent passing of vehicles. We vary the decimation rate from once every 10 seconds to 500 seconds. The memory usage in terms of average cells per set is listed as a function of decimation rate in Table 10, along with their respective segmentation images shown in Figure 13.

Table 10: Decimation statistics

	Decimation Rate (secs)		
	10	**100**	**500**
Docks	1.27	1.40	2.26
Traffic	1.46	1.52	2.29
Science	1.31	1.36	1.67

(a) Average Cells per Set

	Decimation Rate (secs)		
	10	**100**	**500**
Docks	2	4	6
Traffic	2	4	8
Science	3	5	8

(b) Max Cells per Set

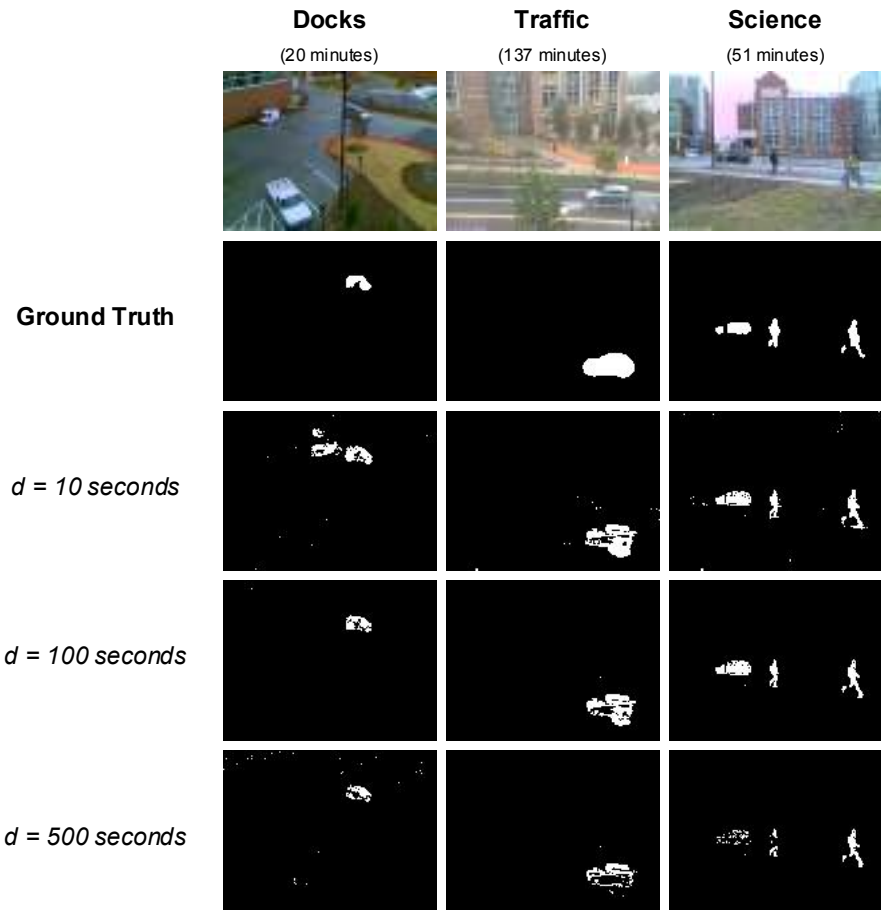

Figure 13: Foreground segmentation as a function of decimation (d)

Table 10(a) represents the memory usage from a global perspective whereas Table 10(b) gives a pixel-level representation. The most frequent decimation rate (10 seconds) has the lowest number of cells per set due to its frequent deletion of cells. At this decimation rate, the memory usage among the sequences is similar. However, the amount by which usage increases with decimation rate differs depending on the foreground activity and illumination conditions within the scene. For example, the *Docks* sequence has sparse foreground activity throughout its duration. As a consequence, it has the lowest "max cells per set" count. In contrast, the traffic sequence has frequent passing vehicles, which results in more cells per set being stored to model different car colors. Even though the *Docks* sequence had much less foreground activity than the *Science* sequence, it had more

cells per set because there were more lighting changes that were kept in the model when using the less frequent decimation rates.

The *Docks* sequence at a decimation rate of 10 seconds shows the unwanted effects that come from setting the decimation rate too frequent. In the circumstance of brief occlusions of background with passing foreground, it is expected that the background will be "remembered" once the foreground object passes. The overly aggressive deletion of cells leads to more false positives because the model "relearned" previously seen background cells that were prematurely deleted. 20 minutes into the *Docks* sequence a grey car enters from the right, drives up the road and briefly stops to drop off a passenger. The car backs up and turns around to exit, using the same path it entered. While this is happening the background model adds cells to the set and updates its observation count to include information about the semi-stationary car during the two times it stops. Figure 14 illustrates the exact path of the vehicle and the time it lingers at each point. The car is in the scene for a total of 37 seconds.

Figure 14: Path of vehicle as it makes two brief stops

This ghosting effect is visible when the decimation rate is 10, which is below the recommended minimum that states $d \geq 2Fth$. In the case of an extended foreground occlusion, the user decides the time-period in which an occluded background will be remembered.

When the decimation rate is infrequent, there is the potential for aliasing of new foreground with previously seen foreground cells that were not deleted from the model. Also, it takes longer for cells that have surpassed the cell threshold in observation count to be deleted from the model. Short-term foreground that was observed just frequently enough to surpass the cell threshold in observation count will remain in the model until the next decimation cycle. In the *Traffic* and *Science* sequences there are more observable false negatives at a decimation rate of 500 seconds than 100 seconds because many cars of similar colors have driven across the road and their cell records were not deleted from the model. This effect was not visible in the *Docks* sequence because there were few transient foreground objects in the scene, making the potential for misclassifications unlikely. The value of the cell threshold can be increased to avoid this if it is critical to application performance.

Runtime Comparison

The MMean algorithm was evaluated against popular backgrounding techniques such as frame differencing, approximated median, sliding window median and mean, weighted mean, and Mixture of Gaussians (MoG) on the eBox-2300 embedded platform (Apewokin and Valentine et al.) [1],[4] shown in Figure 15.

Figure 15: eBox 2300 VESAPC embedded platform

It weighs 505g and its dimensions are 115×115×35mm. Its small size and weight make it amenable for deployment in numerous locations, as it can be easily mounted to walls, monitors, and other fixtures. It operates using a Vortex86 SoC-200MHz fanless CPU with 128MB of onboard SDRAM, having a low power consumption of 15 Watts [16]. To map the MMean algorithm to the eBox-2300 platform, the number of available cells per set was restricted to four, and the model was run using the modified cell structure in Figure 16.

S_r	S_g	S_b	C	Rcount1	Rcount2

Figure 16: Multimodal mean cell structure with recency counters

The recency counters *Rcount1* and *Rcount2* are necessary in this mapping to help chose a replacement cell if a new pixel is seen and its existing count of cells in the set are maxed out. *Rcount1* records how often the cell was observed within a recent time window. *Rcount1* is reset to zero every w frames (here $w = 32$). Upon reset, the value of *Rcount1* is copied over to *Rcount2* so that recency information is not completely lost. When a new cell needs to be created, and all available cells are currently being used, a decision must be made on which cell to replace. The cell to be replaced is selected from the subset of cells seen least recently with the smallest overall count. In the unlikely event that all cells

are observed equally often over an entire window, then the cell with lowest *Count* is replaced.

Figure 17 shows the test sequences used and their output foreground frames produced by the different background modeling techniques with the parameter settings listed in Table 11.

Table 11: Model parameters

Algorithm	**Parameters**
Mean/Median (SW)	|window| = 4
Weighted Mean	$\alpha = 0.1$ for $\mu_t = (1 - \alpha) \times \mu_{t-1} + \alpha x_t$
Mixture of Gaussians (MoG)	K =4 modes, initial weight 0.02, $\alpha = 0.01$, weight threshold T = 0.85
Multimodal Mean	K = 4, $E_x = 30$ for $x \in \{R,G,B\}$, d = 400, w = 32

Figure 18 compares the accuracy of each technique in the form of total false positive and negative errors from each sequence.

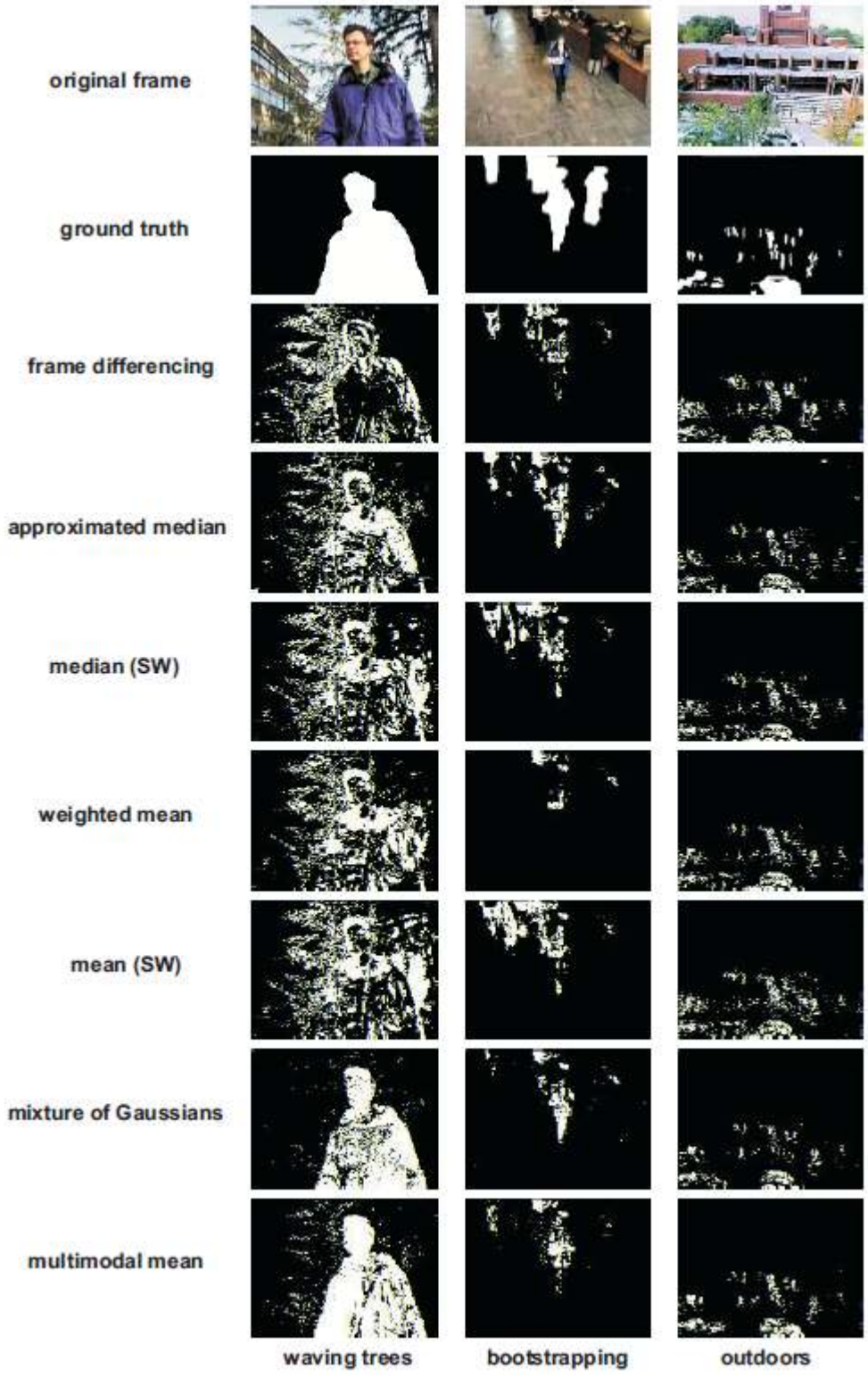

Figure 17: Image quality comparison of background modeling techniques

The runtime comparison is presented in Table 12, which shows the average processing times per frame, average frame rates, and storage requirements for each background modeling method executing on the eBox test platform.

Table 12: Comparison of background modeling techniques on eBox-2300 test platform

Algorithm	Time (ms)	Rate (fps)	Storage (words/pixel)
Frame Differencing	7.6	132.0	1: packed RGB
Approximated Median	8.5	117.3	1: packed RGB
Median (SW)	69.2	14.4	3: 3 char × 4
Weighted Mean	26.8	37.3	1: packed RGB
Mean (SW)	28.2	35.5	3: 3 char × 4
MoG	273.6	3.7	22: 5 FP × 4 modes + 2 int
Multimodal Mean	43.9	22.8	18: (4 int + 2 char) × 4 cells

Notably, MMean provided a 6x improvement in execution time over MoG on the eBox-2300, while using 18% less storage.

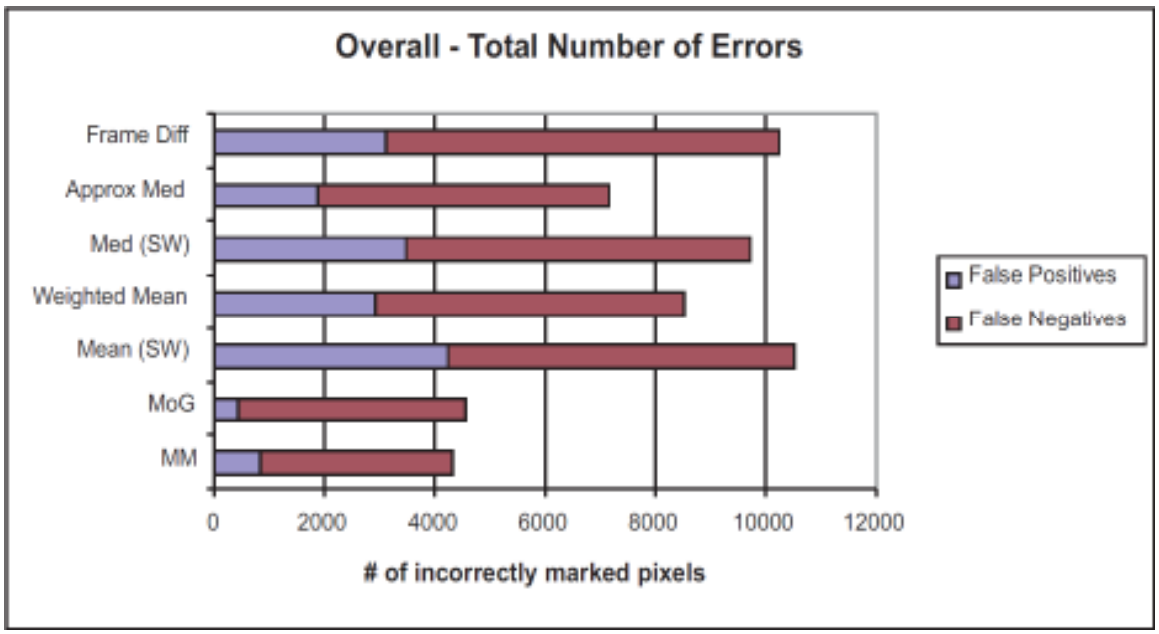

Figure 18: Error comparison between background modeling techniques

It is clear from these results that the MMean technique produces comparable results as MoG at a substantially reduce computational load.

CHAPTER 3 : AN EFFICIENT, CHROMATIC CLUSTERING-BASED BACKGROUND MODEL FOR EMBEDDED VISION PLATFORMS

Introduction

Advances in surveillance technology (low-cost high resolution imagers) and processor architectures (energy-efficient, embedded processors) have enabled a new generation of low-cost embedded surveillance systems. However, new backgrounding techniques are needed to achieve real-time performance within the processing and memory constraints of embedded systems. Most surveillance applications analyze salient foreground objects to recognize specific activities of interest. Foreground is detected by comparing pixels in a scene frame to a reference (background) model. Non-matching pixels are declared as foreground. Adaptive pixel-based approaches statistically update their models to reflect foreground changes (e.g., occlusions) and background changes (e.g., illumination changes) to the scene. Multimodal background models include multiple values for each pixel to better capture dynamic background. These models require significant computation and storage to iterate through and update candidates in the background model. While this improves foreground detection accuracy, it has often prevented an efficient embedded system realization. The multimodal mean background model, for example (Apewokin and Valentine et al.) [1],[4] employs an adaptive, multimodal background framework that efficiently matches pixels against a multiple value set. But with ever increasing imager resolutions, this technique can still struggle to meet required frame rates.

This chapter presents a background modeling optimization approach that reduces processing and storage requirements while maintaining accuracy. A hybrid background

model is presented that exploits the stability of large, persistent background objects so that unnecessary model updates and adaptations are suppressed. These stable object types, such as buildings and roads, usually contain homogenous color distributions that can be identified by analyzing the color composition of a single image frame. Based on the chromatic distribution, a small color palette is defined against which large, permanent background object colors can be compared. During background modeling, the pixels belonging to the permanent background colors are classified as *stable*, while others are declared *adaptable*. A hybrid background segmentation technique is used to process the two classes of pixels. The ``stable'' pixel classification allows quick identification and processing by using the palette as a background color look-up table. Experiments show that the stable class represents 46% to 70% of pixels in our test sequences. The palette is recomputed, as needed, to adapt to gradual, but significant, scene changes (Valentine et al.) [42],[43]. The ``adaptable'' pixel classification invokes the MMean algorithm (Apewokin and Valentine et al.) [1],[4], an adaptive multimodal background model. A set of ratiometric values is maintained for each pixel, allowing dynamic background modeling and value management based on match frequency.

To evaluate the performance of this technique on an embedded platform, it is implemented on an eBox-2300. Our test sequences used for experimentation are shown in Figure 19.

Figure 19: Description of hybrid model test sequences

Scenes *Walkway*, *Sidewalk*, and *Light Post* contain a steady flow of pedestrian traffic moving orthogonal to the camera view. Additionally, scene *Walkway* contain multimodal rustling tree leaves. The sequence named *Courtyard* is captured from a high vantage point, with moving clouds and drastically fluctuating daytime lighting changes. The *CPETS_S7_T6_B* and *i-Lids AVSS Easy* sequences have a significant amount of pedestrian traffic walking toward, parallel, and away from the camera. The

CPETS_S7_T6_B [32] and *i-Lids AVSS Easy* [23] are standard test sequences used in abandoned baggage detection. They have been downsampled in frame rate to *1 fps* for processing on the eBox-2300. The other sequences vary in sampling frame rates between *1 fps* and *10 fps*. The *Walkway*, *Sidewalk*, and *Light Post* sequences were originally collected at Georgia Tech for a pedestrian tracking and counting application.

The presented hybrid background model works well in these real-world, indoor and outdoor sequences. Executing at the required frame rate on the eBox-2300, this model achieves an average speedup of 45% compared to the multimodal mean algorithm alone. It also reduces memory usage by an average of 58%. These improvements support an efficient embedded surveillance system implementation.

Related Work

Typical background modeling techniques continuously update data for each pixel location. Adapting these models involves repeated updating of weights, Gaussians, variances, means, or observation counts. Given the large model size, which is sometimes over three times the storage of an uncompressed image, this requires an enormous number of memory accesses and maintenance operations. Every pixel incurs similar processing overhead, whether or not they represent significant changes in scene content. In contrast to these techniques, we are taking a novel approach by treating certain types of background pixels differently for more efficient processing and storage on embedded processing targets. Specifically, we represent long-term, stable background data compactly using a small palette of 15 colors that are referenced as a look-up table. The color palette, created using a data clustering method to identify chromatically clustered

colors belonging to large, stable background objects, allows us to compactly represent stable pixels with a spatial index map and quickly process them.

Traditional clustering techniques represent unsupervised classification of patterns into distinct groups [25]. They have been used primarily to perform image quantization and image segmentation. Examples of clustering techniques include histogram population, median-cut, octrees, mean-shift segmentation, normalized cuts, and k-means clustering.

Creating an RGB color histogram is one of the simplest ways to cluster color information. In a three-dimensional RGB color histogram, the color space is divided into equally spaced cubes (bins). As the image is scanned, the bin counts are incremented for each matching pixel. This is fast, as only a single scan through the image is required. This technique is commonly referred to as the *Histogram Population Method* [25].

The *Median-Cut Clustering* algorithm takes an RGB color cube and segments it into distinct subsets of smaller cubes. Each box represents a color bin that signifies how many pixels in the image have that color. The boxes are continually subdivided until the number of requested bins are reached. When segmenting the boxes, the split is made along the longest component side such that the two smaller boxes have an approximately equal number of pixels [22].

Octree is a clustering technique that categorizes data within hierarchical trees. Its unique feature is that it clusters based on the bit positions of the data's numerical values. When used to cluster RGB color data, the tree hierarchy is based on the eight bit-positions of a pixel's RGB values. Indexing into the octree is performed by grouping bit positions of the pixel's RGB color components into three-bit words. The top level

contains eight child branches. Each branch can contain a leaf, or eight additional child branches. Color clustering is performed by scanning the image with each pixel being inserted into its appropriate leaf in the octree. A leaf represents all the pixel values that fall within the bit index ranges ascribed by the tree. The lowest level of the octree, indexed by the LSBs of the RGB pixel value, contains the most detailed grouping of colors [19].

Mean shift segmentation [18] and *normalized cuts* in graph-theoretic segmentation [26] are often used for image segmentation. In the mean shift segmentation algorithm, a number of initial search window locations are first chosen uniformly in the color space. The centroids of the data points within each window are computed. Afterwards, the windows are shifted until their centers are matching the computed centroids, and the windows no longer move. Overlapping windows are usually merged. Normalized cuts in graph theoretic segmentation is more complex, as it involves creating a weighted, undirected graph from the pixel data points. The edges that are formed between every pair of pixel points are a function of similarity between the pair. In clustering, the set of points are partitioned into disjoint sets such that a normalized sum of the weights of all the edges connecting a pair of partitions is minimized.

K-means analysis is a technique used to optimize a set of existing cluster centroids. In this algorithm, k number of centroids are defined a priori. Each pixel in the image is matched with its nearest centroid, creating a set of k clusters. Following the cluster creation, barycenters of each cluster are computed, resulting in an updated set of centroids. Each pixel is then re-associated to the new set of centroids. This process is repeated until the centroids no longer move in subsequent steps. Once this stage is

achieved, an optimal set of cluster centroids are found. This set is ultimately dependent on the initial quality of starting centroids, thus the optimized result may only be locally-optimal [29].

A critical difference in our goals compared with traditional clustering methods is that we are interested in identifying and modeling stable regions of pixels, which do not completely cover the image. Traditional methods aim to cluster the entire image such that each segment is mapped to a visually optimal cluster color [37].

Chromatic distribution analysis for background color palette creation

Our approach to efficient background modeling is based on the idea that we can treat certain background features as salient, rather than something to ignore and subtract from the image. This ``background saliency'' is defined by physically large and stable objects, having similarly large clusters of chromaticity. Examples of these include walls, roads, sidewalks, lawns, and buildings. Furthermore, since these objects are large, relative to their field of view in the image frame, we can find their basic color composition by determining the most popular colors in the image frame. This is conceptually illustrated in Figure 20, where we have outlined in each frame examples of two large, stable background objects (grassy regions, a building's wall, and a sidewalk).

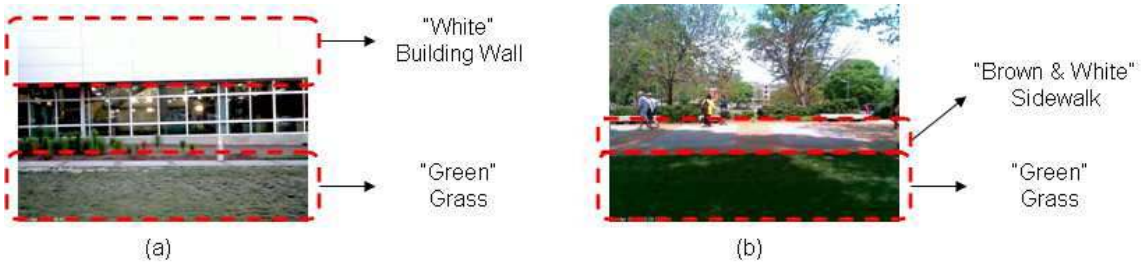

Figure 20: Conceptual highlighting of large, stable background regions in common outdoor settings

By identifying these regions through chromatic distribution analysis, we can quickly test pixels in these regions to detect changes due to foreground activity using a storage-efficient background model.

The background model for these stable regions consists of a color palette representing the 15 most popular colors in the image. Three key requirements determined our choice of color distribution analysis algorithm to generate the color palette. The first is that the analysis should consistently select colors that are in fact popular and that occur in regions that correspond to what a person would identify as stable background regions. The second is temporal stability of pixel classifications and palette colors: if a pixel is classified as stable, it should match to its assigned palette color (which we refer to as its ``brand'') consistently over a long period of time. The average deviation over time of the pixel value from the brand palette color to which the pixel is assigned should be small. In addition, the percentage of stable pixels in the image that consistently match their assigned brand over time should be high. The third requirement is that the palette should be efficient to compute and store, since it may need to be recomputed occasionally due to significant changes in the scene, such as large illumination changes or the appearance of new large occlusions.

We experimented with several color clustering algorithms in the RGB colorspace. Two algorithms, median-cut and histogram population, rose to the top of candidate palette generation algorithms in satisfying our requirements.

Histogram Population Method

Our implementation of the histogram population method takes the three-dimensional colorspace of 2^{24} = 16M colors and quantizes each color component axis of

2^8 = 256 color intensity values into 16 bins, each bin having a range of 16 intensity values. This method uses a total of $2^4 \times 2^4 \times 2^4 = 2^{12}$ = 4096 bins. The initialization image is scanned in raster-order. For each RGB color pixel scanned, its associated bin in the histogram is incremented by one. At the end of the scan, a list of 4096 color bins is created and populated with a count of how many pixels in the image belonged to each bin. To create the 15-color palette, the bin list is sorted in descending order, with the bin-coordinate centroid of the top 15 entries assigned to the palette.

Our implementation of the median-cut technique begins with a full-resolution RGB cube containing 2^{24} = 16M color bins. The initialization image is scanned in raster order to populate the bins. After the initial scan the bins are represented as a single cube having diagonally extreme vertices of *<0,0,0>,<255,255,255>*. The color bins can be thought of as a blob in the three-dimensional space, where the cube is split into two along the longest end-to-end component dimension. The split is made such that the two new cubes have an equal amount of color bin counts. For example, the original cube can be split into two such that the new diagonally extreme vertices are *<0,0,0>,<255,255,100>* and *<0,0,101>,<255,255,255>*. The splitting of color cubes continue until 15 cubes are created. The average color of the bins populating each cube becomes an entry into the palette.

We evaluated the histogram population and median-cut method on initialization frames from all our test sequences. Our tests compared their determination of the most popular colors and how well the selections held over time. Results of this experiment are shown in Figure 21.

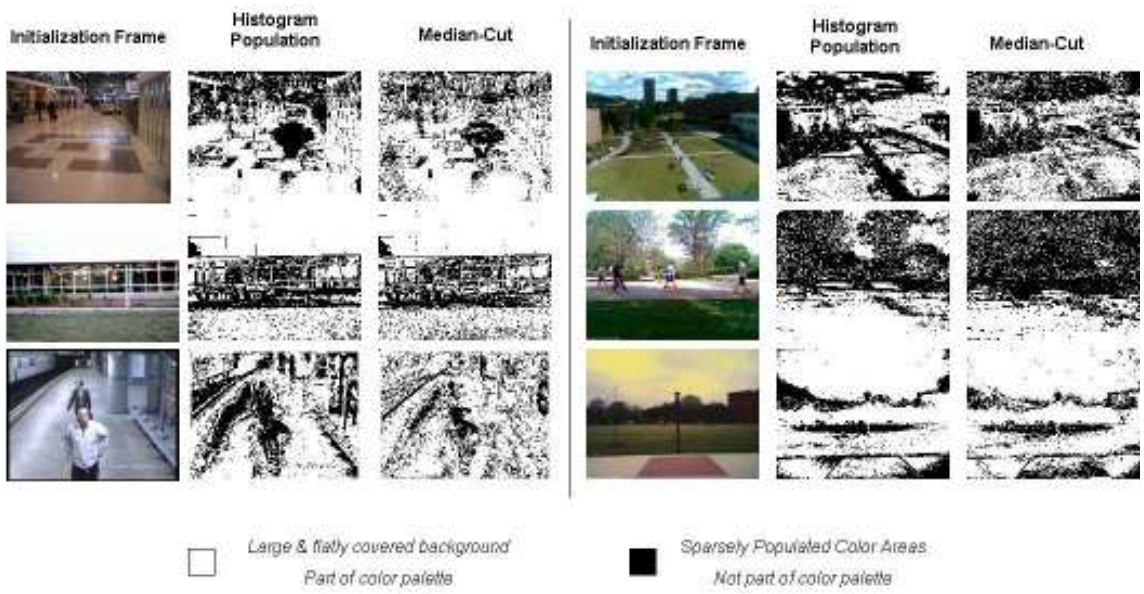

Figure 21: Comparison of popular color selection using histogram population and median-cut

The black areas in Figure 21 are the locations where no palette color has been assigned. We expect these to be in places with occluding foreground or smaller-sized background objects and that expectation is met. The white places in the image are areas where a palette color was found. Visually, the resulting palletized images do not differ substantially between the two clustering algorithms. They pick generally the same colors in similar spatial regions, which correspond to humanly perceived stable background regions.

To test the temporal stability of the color palette generation methods, we used the following two metrics. The first metric focuses on pixels classified as stable in the image; these are pixels in the initial reference image from which the palette is generated that match within a threshold T_1 of one of the palette colors, i.e., its brand. A palette color is a centroid representing a cluster of color intensities near it in a three-dimensional colorspace. When branding pixels in the current frame, they should be reflective of the centroid and not simply an intensity near the centroid. In our experiments, T_1 is chosen to

be 10 to ensure each pixel is branded to a palette color that is in near proximity to the pixel's current color. In this experiment, we compute the average sum of absolute difference (*SAD*) between each stable pixel's brand and the pixel's color value in subsequent frames. The *SAD* between the current pixel and its palette color is totaled, with the average recorded for all pixels. It is desirable that the *SAD* remain steady over time, since a diverging increase of *SAD* over time would indicate temporal instability in the color distribution.

The left plot of Figure 22 shows an average of the results using this metric on our six test sequences.

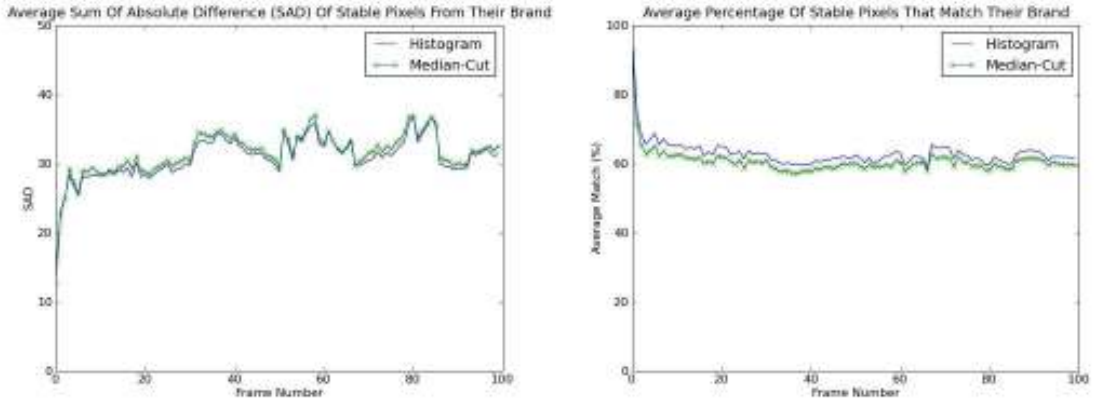

Figure 22: Evaluating the temporal stability of palette colors (left) and pixel classification (right)

The steady level of the plot illustrates how well the palette holds over time, under stable scene conditions (i.e. no lighting fluctuations or large occlusions). It illustrates that the palette colors generated by both methods remain consistent, with an average *SAD* of 30 to 40 between the stable branded pixels in the image frame and their mapped palette color. The second metric we measured is the percentage of pixels classified as stable that match their assigned brand within a threshold T_2. In our experiments, $T_2 = 30$ to account for the standard deviation of palette colors. The result of applying this metric is shown in right

plot of Figure 22. It indicates that approximately 60% of the palletized locations (i.e., branded, stable pixels) were able to be matched over a period of 100 frames. These measurements constitute an average of the total runs across the six test sequences from Figure 19.

Although efficient, the median-cut technique requires repetitive analysis and splitting of color cubes to reach the number of needed colors. In contrast, the histogram population technique is a more streamlined composition of linear bin population, followed by a simple list sort. The runtime performance difference between the two palette generation methods is not substantial when amortized across the runtime of modeling the entire sequence. However, since the two techniques produce similar palette colors in the same regions, with high temporal stability, we chose the histogram population method on which to base our fast, palette-based background model because of its inherent simplicity which yields slightly faster performance.

Fast, hybrid background modeling using color palette and MMean

Our fast, hybrid background modeling algorithm consists of three main phases: color palette-based background model initialization based on chromatic cluster analysis, hybrid background model matching (palette-based for stable regions and multimodal mean for adaptable regions), and palette recomputation when needed.

The palette-based background model that is used to efficiently match and process stable background is initialized in two steps: color palette generation and spatial index map computation. First, a color palette is created based on a reference image of the sequence. In general, the reference image can be generated from a training period in which a more expensive multimodal background modeling technique, such as MoG or

MMean is used to compute the reference image. However, for efficient on-line palette computation (and later recomputation on the fly), the experiments in this paper use the first image of the sequence as the reference. Chromatic cluster analysis is performed by applying the histogram population method to the reference image. The top k color bins resulting from the histogram analysis become palette colors.

The second step of model initialization creates a spatial index map S from the reference image F. The spatial index map has an entry for each pixel in image F: if the pixel matches one of the palette colors within a predefined threshold T_1, the corresponding location in the spatial index map S_x is assigned the index of the color in the palette; otherwise it is assigned a null entry (or value 0). The spatial index map efficiently encodes pixel classifications as either stable (assigned index into the palette) or adaptable (assigned 0). More precisely, each pixel F_x at location x in the reference image F, is compared to each of the k palette colors. If there is a palette color P_m at palette index m in $\{1, 2, ..., k\}$ such that Equation 5 is satisfied for each color component j:

$$\left| F_{x,j} - P_{m,j} \right| \leq T_1, \forall j \in \{R, G, B\} \tag{5}$$

then the spatial index map S_x at location x is assigned palette index m of the matching palette color; otherwise it is assigned 0.

$$S_x = \begin{cases} m \;\; if \;\; \exists m \in \{1,2,...,k\} \; s.t. \;\; Equation(5) \, holds \\ 0 \quad otherwise \end{cases} \tag{6}$$

(In this notation, $F_{x,j}$ is the reference image pixel value for color component j at location x, and $P_{m,j}$ is the value of the color component j for palette index m.)

For computational efficiency, we assume each color component has the same standard deviation, so that we can use a single threshold value across all components. In the experiments in this paper, we bound the threshold to a component deviation of $T_1 = 10$ to ensure that only pixels in the image closely matching the palette colors are assigned entries in the spatial index map. The structure of the palette is shown in Figure 23.

Observation Percentage		
R_1	G_1	B_1
	.	
	.	
	.	
	.	
R_k	G_k	B_k

Figure 23: Color palette structure

The palette contains the RGB components of each clustered color at list index *i*, and the total percentage of pixels classified as matching a palette color (or stable) in the image. The palette is small, requiring only [(3 × k) + 1] bytes of storage. The palette and spatial index map serve as a check for stable background pixels. Only a simple comparison of the current pixel color and the color of its assigned palette entry is needed to distinguish between foreground and background. This reduces computations related to model adaptations, as well as reducing the number of accesses to system memory.

The hybrid background model matching algorithm works as follows. For each pixel I_x in the current image frame at location *x*, the spatial index map for location *x* is looked up: $m = S_x$. If *m* is 0, the pixel is treated as adaptable and processed using the multimodal mean algorithm described in Chapter 2. Otherwise, the pixel is treated as stable and the index *m* is used to look up a palette color P_m. If the palette color matches the current pixel value within a threshold T_2 for each color component (according to Equation 7), the pixel is treated as background, otherwise it is declared foreground.

$$\left| I_{x,j} - P_{m,j} \right| \leq T_2, \forall j \in \{R, G, B\} \quad (7)$$

In our experiments, T_2 is chosen to be 30 to accommodate the standard deviations of the palette colors.

The MMean algorithm is an efficient multimodal modeling approach for pixel locations classified as adaptable (i.e., having a null palette entry). The current pixel value is compared against a set of background cells representing observed background modes for that location. Each cell contains a running sum of observed RGB component values, an observation count, and two recency counts, as was illustrated in Figure 16 from Chapter 2. The palette-based approach reduces both the computational overhead and memory access overhead of the MMean algorithm. Computationally, it saves at least five additions per pixel that are used to update color sums and observation and recency counts, and three divisions to compute the background color average. In terms of memory accesses, it saves at least four loads for the color sums and observation count, and four stores after they have been updated. The savings in memory usage from the palette approach is significant as memory access latencies hinder runtime performance more prevalently in embedded systems, such as the eBox-2300.

Over time, dramatic changes in lighting can alter the general color distribution of the scene. In such events, the chromatic distribution needs to be re-evaluated to align with current conditions. To determine when this is necessary, we track the percentage of successfully matched stable background pixels identified by the palette for each frame. A re-initialization of the palette is triggered when the observed coverage consistently falls below its historically measured match percentage. More specifically, the *match*

percentage, $MatP_t$, tracks the value of the highest observed match percentage of t processed frames:

$$MatP_t = \max(ObsP_i) \quad for\{i = 0,...,t-1\} \tag{8}$$

where $ObsP_i$ is the percentage of pixels in the current frame i that match their associated color in the spatial index map. When the scene is in steady state, there are no large deviations in the observed match percentage. A match counter, called *MatCount*, is a scene-wide counter that is incremented by one every time the current frame's observed match percentage Obs_t falls a significant amount *Pdev* below the steady state value $MatP_t$. If no significant deviation from steady state is observed in a subsequent frame, then *MatCount* is decremented by one but saturated at zero.

$$MatCount_i = \begin{cases} MatCount_{t-1} + 1 & if\ \ MatP_t - ObsP_t \geq Pdev \\ \max(0, MatCount_{t-1} - 1) & otherwise \end{cases} \tag{9}$$

Ideally, the observed percentage of stable branded pixels should be the same as its historical percentage. The amount of deviation between the observed percentage and historical percentage is dependent upon the source of the lighting change and the areas of the scene it affects. We set *Pdev* = 5% so that the palette can be recomputed due to global and local lighting changes. This operates under the assumption that no more than five percent of the stable branded pixels will be occluded by foreground for a sustained period of time.

If *MatCount* becomes too high (above a threshold *Pth*), a significant, and persistent change has occurred in the scene. When this happens the palette is recomputed. A situation that should be avoided is recomputing the palette for a temporary change, or recomputing the palette in the middle of a transition to a new set of lighting conditions.

The duration of the transition, and if the resulting change is persistent, cannot be known ahead of time. The value of *Pth* is set based on the user's expectation of how long the model should wait for the original palette to reassert itself before making the decision to recompute. In our experiments, we expect that if the new lighting conditions exist persistently for more than 10 seconds then the palette should be recomputed to reflect this. We use *Pth = 10* so that short-term fluctuations (one to two seconds) do not factor into the recomputation decision, and to ensure that there has been at least 10 seconds worth of observed changes in scene conditions. *Pth* can be tightened in scenes with more controlled lighting, such as indoor locations, or widened based on the expected transition periods between persistent environmental states.

The Courtyard scene has several large, persistent illumination changes caused by varying cloud cover. In several places within the sequence, the lighting switches from dark to light settings within the span of a few seconds. Figure 24 shows a plot of the percentage of palletized pixels detected over the entire duration of the Courtyard sequence.

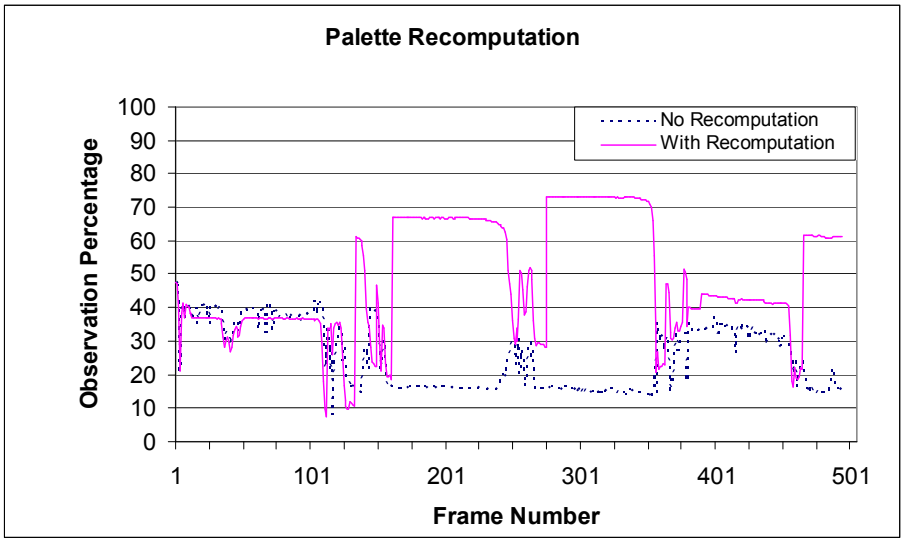

Figure 24: Palette recomputation to lighting fluctuations in the Courtyard sequence

The dashed line plot shows the matching percentage using the initial color palette with no recomputation. The solid lined plot shows the matching percentage with the palette that has been recomputed after tracking the changes in observation percentage using *MatCount*. The unmodified palette match percentage falls to below 40%, resulting in fewer pixels being processed in the fast, stable pixel classification. Recomputing the palette to the current lighting conditions allows a far greater number of pixels to be processed as stable pixels, up to 73%, resulting in faster run-time execution.

For bandwidth purposes, it is common for video cameras to capture raw image data from the sensor and encode the video as an MPEG stream before sending it to the host for further processing. The reference frames from the MPEG stream are typically what is used to represent the still video image that is seen on the host. While the MPEG standard uses motion vectors to temporally compress the video stream, this information would not be applicable for palette generation. Since many commercial-off-the-shelf cameras feature automatic gain control and white balancing, having a way to synchronize palette re-initialization along with such camera controls is preferable.

Evaluation and Results

We have evaluated the performance of our hybrid background model in terms of processing frame rates and segmentation accuracy (false positives and false negatives). We chose the eBox 2300 PC running Windows Embedded CE 6.0 as a testing platform because its moderate specifications are representative of the algorithm design constraints that developers will encounter when working with portable, embedded multimedia architectures. It uses a low-power 200MHz x86 processor on a fanless Vortex86 SoC.

Our code was developed entirely in C and compiled using the Microsoft Visual C++ 2005 Embedded Platform Builder. We compiled a lightweight kernel of size 19MB to run our hybrid background model on the eBox. Test JPEG image sequences are preloaded into memory prior to algorithm execution.

Runtime Performance

We have measured the run-time performance of our hybrid background modeling algorithm on the eBox 2300 embedded PC platform, using the diverse set of test sequences listed in Table 13.

Table 13: Test sequences used to evaluate hybrid model

Sequence	Number of Frames	Ground Truth Frame
CPETS_S7_T6_B	137	52
Sidewalk	600	85
AVSS AB EASY	118	55
Courtyard	499	427
Walkway	749	63
Light Post	153	95

All sequences have a frame size of 160×120, having a starting frame number of zero. The resolution of the image sequences was downsampled to 160×120 due to the limited amount of available onboard SDRAM of the eBox. Table 14 lists the algorithm parameters that were used in our experiments.

Table 14: Hybrid model parameters

Hybrid Model Parameters
4 modes (MMean), Fth = 10
15 Palette Colors
T_1 = 10, T_2 = 30
Pth = 10, Pdev = 5%

Noise occurs in the foreground frame due to errors, associated with stable branded background pixels that temporally exceed the expected standard deviation (T_2) of the palette color, gradual changes in lighting, and misclassified pixels in the multimodal-branded regions (Choi and Valentine et al.) [12]. We apply vertical and horizontal morphological filters to the foreground segmentation to clean up these false positives. To conserve memory usage on the eBox-2300 a 16-bit word is used as sliding window for the horizontal and vertical directions of the morphological operators. The 16-bit word stores the binary output of the foreground segmentation for 16 neighboring pixels in each of its bits. As the foreground image is scanned, the oldest pixel in the 16-bit word is discarded with an arithmetic right shift, and the current pixel's foreground/background state is placed into the least significant bit of the word with an arithmetic OR operation. The total sum of the pixel states in the 16-bit word is continually maintained by adding the new state and subtracting the old state after each arithmetic shift and OR operation. A pixel that was declared as foreground is reclassified as background if the total sum of its neighbors in the buffer is less than four pixels.

In previous experiments described in Chapter 2, the MMean was shown to perform 6 times faster than MoG using 18% less storage. On average, we have achieved a 45% speedup over the MMean in frame rate using the hybrid model, as shown in Figure 25.

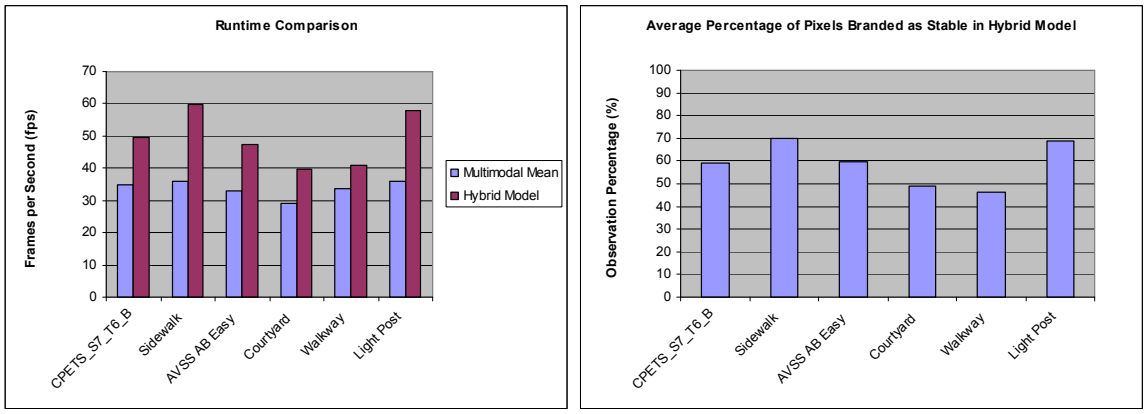

Figure 25: Comparison of multimodal mean and hybrid model performance on eBox test platform in frames per second (fps)

This is accomplished through the reduced number of memory accesses and simplified model operations of the palette-based approach. Table 15 shows the storage requirements of the two approaches, where λ represents the percentage of pixels in the image frame that have been branded as stable.

Table 15: Hybrid model storage requirements comparison

Method	Storage (32-bit words/frame)
Multimodal Mean	18×Width×Height
Palette Spatial Index Map	0.25×Width×Height
Hybrid Model	(Width×Height)[(0.25λ) + 18(1-λ)]

The palette approach requires only an index entry for each pixel in the image that was branded as stable whereas the MMean needs storage space for observation counts and component sums, where 18 words is derived from four ints (three color sums and observation count), two chars (two recency counts), times a maximum of four cells per

pixel. The palette model requires 0.25 words per pixel to store the palette table index. When combined with the morphological operators, this manifests itself during runtime as an average 3% increase of data load operations and a 39% reduction in data storage operations. The analysis was done by recording the number of memory load operations used to access data from the current image frame, spatial index map, palette, and MMean cells, and the number of memory storage operations used to write the output frame and update MMean cells. Figure 26 details the number of memory operations executed for each test sequence using the two approaches. The number of load operations are comparable as they both need to retrieve the current image pixel and a reference pixel for matching.

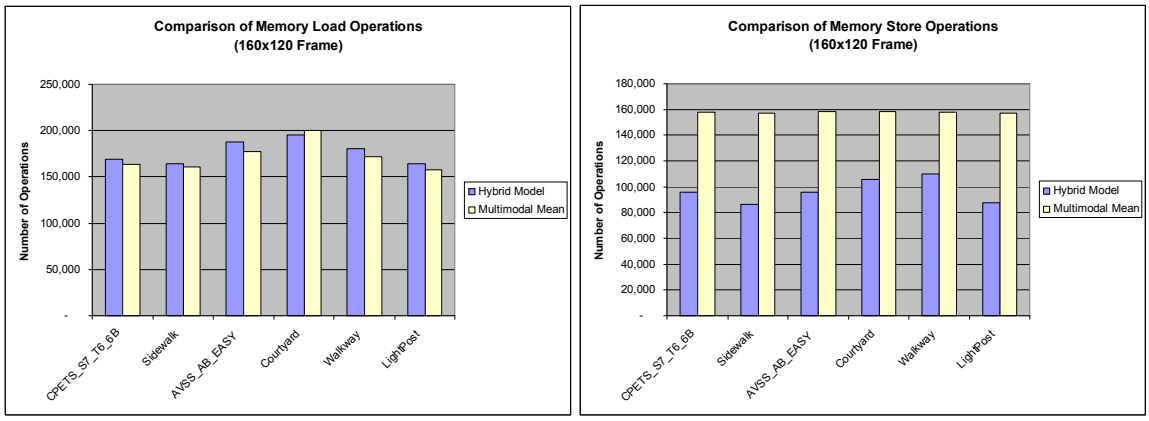

Figure 26: Runtime memory operations comparison between hybrid and multimodal mean models

The hybrid model has additional load operations because of the extra foreground image data loads for the vertical morphological operator. In contrast, the two methods differ substantially in storage operations as the MMean requires additional operations to update observation counts, color sums, and perform cell replacements. On average, 58.9% of the pixels are branded as stable. This results in an average storage savings of 58%.

The improvement in runtime performance on the eBox is proportional to the coverage percentage of stable branded pixels in the sequence. For example, the *Sidewalk* and *Light Post* sequences feature highly stable background features such as the building walls, sidewalks, and grass. The large, palletized background regions in these sequences cover 60% the entire image frame, resulting in large performance boosts over the *Walkway* sequence, which has a significant amount of dynamic background elements (rustling tree leaves) that are processed in the slower, adaptable classification by the MMean. The average cost of computing the palette and spatial index map is 84 ms whereas the average cost to compute a foreground frame is 21 ms. Therefore, if the palette is computed a few times over the duration of the sequence, it will have little impact on overall runtime. However, consistent and frequent palette recomputations, i.e. more than once every 50 frames, considerably reduces the processing frame rate.

Hybrid Model Accuracy

Our algorithm is able to maintain reasonable accuracy in foreground detection even though a significant percentage of pixels are processed in a coarse manner. Exact false positive and negative measurements for the hybrid background model and stand-alone MMean are shown in Figure 27.

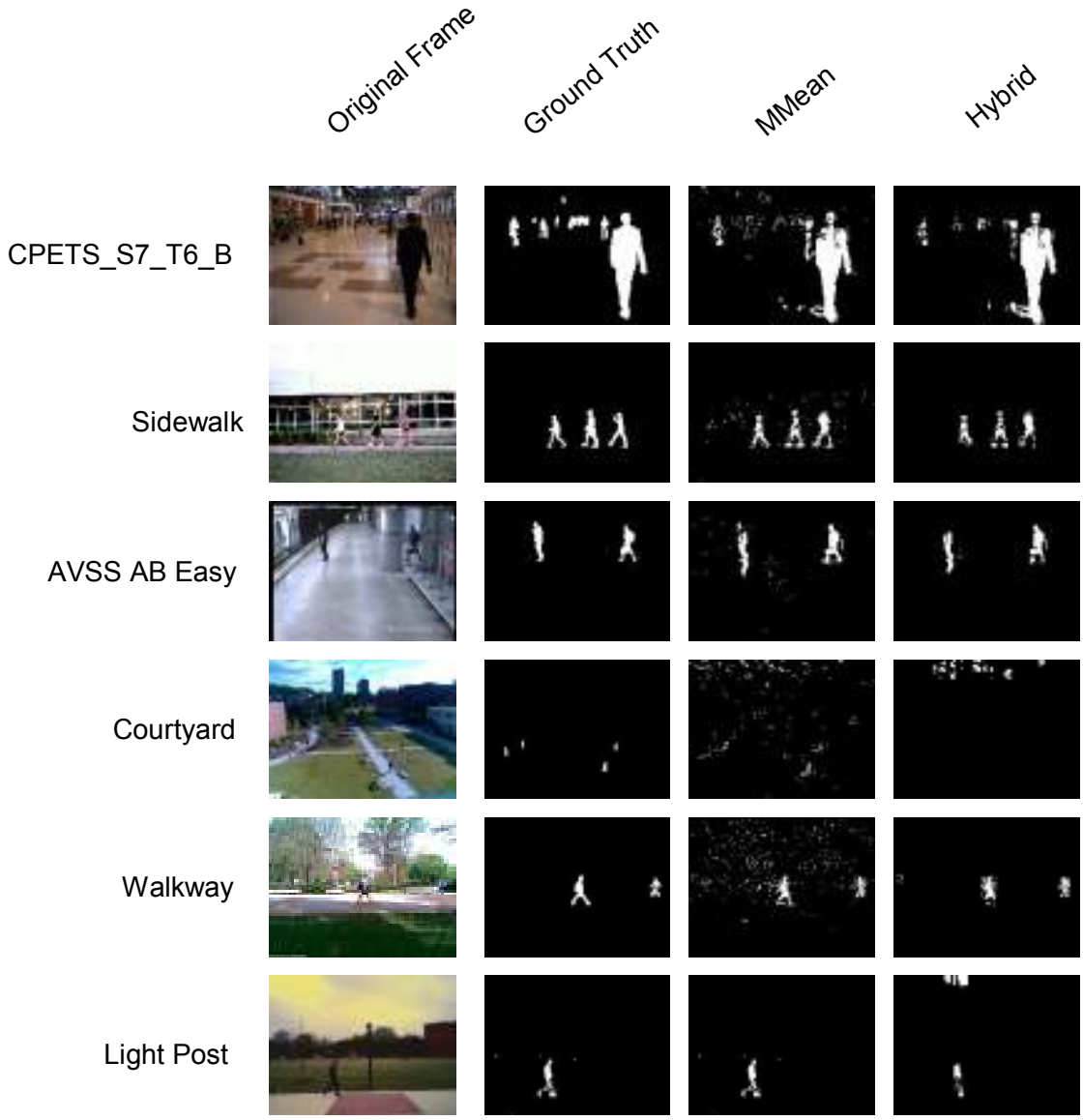

Figure 27: Hybrid model foreground detection image comparison

Figure 25 lists the average percentage of pixels that matched their pre-assigned palette color over the duration of the scene. It also compares the frame rates of the hybrid model with that of the stand-alone MMean. The increased frame rates achieved when combining the palette-based matching with MMean highlights the execution time improvements that can be gained through suppressed adaptation of long-term scene elements. Depending on

scene activity and environmental conditions (e.g. lighting, weather), the palette was able to match 46% - 70% of the scene pixels over the duration of our test sequences.

Effects of Algorithm Parameter Selection

The hybrid background model relies on five independent parameters: T_1, T_2, *Fth*, *Pdev*, and *Pth*. We run a set of experiments, using different combinations of these parameters, to outline the effects they have on the hybrid model output. The matching threshold T_1 is used only in spatial index map creation to match pixels in the initialization frame to colors in the palette. In contrast, the background matching threshold T_2 is used to detect foreground pixels in each frame of the sequence by comparing them to known background pixels. The foreground threshold *Fth* determines how long a new pixel can be observed as foreground before it is absorbed into the background. *Pdev* is the allowed percentage deviation from the steady coverage percentage of observed, stable branded pixels. *Pth*, used in conjunction with *Pdev*, is a temporal measure of sustained deviation, which triggers palette recomputation when exceeded. Given that there are five modeling parameters, there are several possible combinations for experimentation. In this set of experiments we use the operating point *Pdev* = 5%, *Pth* = 10 frames, *Fth* = 10 frames, T_1 = 10, and T_2 = 30 around which one parameter is varied while the remaining are held constant. This operating point was used to produce the results shown in Figure 27. We present the results in tabular form, with entries corresponding to the operating point highlighted in bold. The dependent parameters are accuracy (in the form of the Jaccard coefficient [38]) and average percentage of observed, stable branded pixels.

The Jaccard coefficient J_c is the ratio of true positives (*TP*) to the sum of false positives (*FP*), false negatives (*FN*), and true positives (*TP*), as shown in Equation 2. We

use the Jaccard coefficient as an accuracy metric because it mitigates the effects of scenes with overwhelmingly large amounts of true negative pixels on reported accuracy, and normalizes the viewing of error pixels across sequences and modeling parameters. Coefficient values close to one are preferable, since they will have a small sum of error pixels. In the results below, the Jaccard coefficient is measured using the ground truth frames presented in Figure 27.

Since the average coverage percentage is related to the runtime speedup when using the hybrid model, higher coverage percentages are preferable, as long as they do not hinder accuracy. The coverage percentages presented below are an average of the observed coverage percentages over the duration of each sequence. T_1 reflects a tradeoff between accuracy and execution time. Table 16 shows the runtime effects when varying T_1 from 8 to 30 for the test sequences.

Table 16: Average coverage percentage as a function of T_1, where T_2 = 30, *Pdev* = 5, *Pth* = 10

T_1	CPETS_S7_T6_B	Sidewalk	AVSS AB Easy	Courtyard	Walkway	Light Post
8	51.76	65.36	49.73	43.38	40.19	59.46
10	**59.07**	**69.98**	**59.82**	**48.93**	**46.09**	**68.95**
12	63.15	72.95	67.61	52.95	49.73	73.88
14	64.81	75.40	69.20	56.54	52.43	77.32
16	67.55	76.63	74.87	57.97	55.85	79.89
30	71.66	81.23	76.65	65.21	64.75	88.65

Increasing T_1 causes higher coverage and thus faster runtimes. However, increasing T_1 too much causes a sharp drop in accuracy, as shown in Table 17.

Table 17: Jaccard coefficient (Jc) as a function of T_1, where T_2 = 30, *Pdev* = 5, *Pth* = 10

T_1	CPETS_S7_T6_B	Sidewalk	AVSS AB Easy	Courtyard	Walkway	Light Post
8	0.74	0.66	0.59	0.22	0.59	0.41
10	**0.73**	**0.67**	**0.66**	**0.19**	**0.58**	**0.46**
12	0.74	0.67	0.51	0.19	0.55	0.48
14	0.68	0.68	0.35	0.13	0.51	0.53
16	0.60	0.68	0.62	0.12	0.56	0.56
30	0.55	0.28	0.20	0.15	0.09	0.36

Increasing T_1 to 30 (equal to T_2) causes ambiguities between stable branded pixels and foreground. Varying T_1 from 8 to 30 causes an average 46% *increase* in coverage percentage, but it causes an average 46% *decrease* in accuracy. We chose our operating point T_1 = 10 to serve as a balance between runtime performance and model accuracy.

Varying T_2 shows the typical effects of fixed thresholding on foreground frames. As seen by the Jaccard coefficients in Table 18, there is a rise, peak, and fall in accuracy as the threshold transitions from narrow to wide in most of the sequences.

Table 18: Jaccard coefficient (Jc) as a function of T_2, where T_1 = 10, *Pdev* = 5, *Pth* = 10

T_2	CPETS_S7_T6_B	Sidewalk	AVSS AB Easy	Courtyard	Walkway	Light Post
20	0.35	0.60	0.57	0.09	0.13	0.31
30	**0.73**	**0.67**	**0.66**	**0.19**	**0.58**	**0.46**
40	0.75	0.63	0.64	0.30	0.60	0.56
50	0.72	0.60	0.36	0.32	0.60	0.50

The narrow thresholds cause more false positives while the wider thresholds will cause more false negatives. We observed that the effect of varying T_2 on the average coverage percentage, and therefore execution time, is low. Our experiments showed that the

average standard deviation of the average coverage percentage was less than two when T_2 is varied around the operating point.

Varying the other three parameters, *Fth*, *Pdev*, and *Pth*, had little effect on accuracy or runtime performance. The average standard deviation of the Jaccard coefficient when *Fth* was varied from 5 to 15, in increments of 5, was 0.02, with no changes in coverage percentage. The purpose of *Pdev* and *Pth* parameters is to trigger a re-initialization of the palette when a significant environmental change has occurred. When varying these two parameters about the operating point their effects on the output were not significant. In some cases, no change in coverage percentage or accuracy occurred. In our experiments *Pdev* was varied from 3% to 9%, in increments of 2%. *Pth* was varied from 5 to 15 frames in increments of 5. The average standard deviation of the Jaccard coefficient at the ground truth frame was 0.04 when varying *Pdev* and *Pth*. The average standard deviation in stable branded coverage percentage was 0.69% when varying *Pdev* and 0.55% when varying *Pth* at the ground truth frame. No significant changes are observed because these two parameters affect the long-term accuracy and runtime of the model. Their short-term effects are negligible since in many cases, re-initialization has corrected the palette by the time the ground truth frame has been reached, or the environmental conditions of the scene are stable enough such that recomputation is not needed.

Table 19 shows the error attributed to the adaptable and stable branded pixels separately at the operating point.

Table 19: Jaccard coefficient (*Jc*) for the stable branded regions and the MMean, where $T_2 = 30$, where $T_1 = 10$, $Pdev = 5$, $Pth = 10$

	CPETS_S7_T6_B	Sidewalk	AVSS AB Easy	Courtyard	Walkway	Light Post
Stable Branded	0.77	0.64	0.70	0.04	0.39	0.38
Adaptable	0.68	0.68	0.58	0.32	0.63	0.69

As expected, in the majority of sequences presented, regions that use the MMean (adaptable regions) are more accurate. However, in some cases the opposite is true due to the environmental characteristics of the specific scenes (i.e., level of foreground occlusion in a particular area or lighting changes). The results presented in behave similarly when other operating points are chosen, with some differences in overall accuracy and runtime depending on how close parameters such as T_1 and T_2 are to each other, where setting T_1 equal to T_2 is not recommended.

CHAPTER 4 : MULTI-SCALE TEMPORAL MODELING FOR MIDGROUND ANALYSIS

Introduction

Most surveillance applications use background modeling to factor out elements of the scene that are stationary or changing in uninteresting ways (such as swaying tree branches or rippling waves). This allows salient objects in the foreground to be more readily monitored, tracked, and analyzed for normal or anomalous behavior. However, many crucial surveillance tasks require attention to new objects that appear and persist over time in the midst of a rapidly changing, cluttered scene (e.g., a suitcase left unattended in a crowded airline terminal for several minutes, a person or vehicle loitering near a busy street). Salient objects in these scenarios do not fit within the typical foreground detection framework prevalent in background models. These objects types lie between foreground and background, in a *midground* realm whose temporal scale is defined by the application.

The human visual system has evolved to detect rapidly moving objects, and is ill-suited to perceiving changes in varied time-scales, making it difficult to detect suspicious activities in dense environments. Automated video surveillance systems hold the potential to "tune in" to these types of changes within a temporal window defined by the application. A suspicious event could be a person loitering in a restricted area or a dangerous object being left unattended in a public area. In this class of threats, saliency is related to the object's temporal properties, i.e. duration of existence in the scene, as opposed to whether it is new or previously seen. The temporal properties that determine saliency are: when an entity appears, when it stops moving, when it starts moving, and

how long it stays stationary or lingering. Traditional binary background/foreground models are unable to characterize this type of saliency.

This chapter introduces a midground detection technique that explicitly models a precisely defined temporal window during which persistent foreground objects are classified as midground (Valentine et al.) [44]. Our technique uses a novel unified model for temporal scene analysis that applies to a wide range of applications having differing temporal constraints. For example, applications such as loitering detection or abandoned object detection define saliency on a different time scale. An abandoned object detection application can operate on the order of seconds whereas a loitering detection application could operate on the order of minutes. When operating in these differing domains, our midground detection approach allows more precision than a bi-state background/foreground model, as its input parameters can be adjusted to fit the temporal constraints of the application.

Midground object detection employs a pixel-level, adaptive multimodal background model, called multimodal mean, which consists of temporal records pertaining to object visibility under occlusions, birth date, and observation length [44]. The model separates scene elements into three categories: long-lived, persistent background elements, short-lived (ephemeral), moving foreground elements, and newly stationary, persistent (non-ephemeral) midground objects. For computational efficiency and high throughput, we use integer arithmetic operations to produce the model. Our results demonstrate a fast and effective adaptive implementation that is able to detect midground objects in real-world scenes.

Related Work

A variety of techniques exist to detect stationary objects. Matthew et al. used time-based pixel history values within the framework of MoG background subtraction to detect static objects [30]. Specifically, they place pixel modes into one of four states, Creation (CR), Foreground Gaussian (FG), Background Gaussian (BG), and Background Dominant Gaussian (BDG). The manner in which the pixel transitions between states is used to detect new stationary objects. If the BDG is different after reordering of Gaussians, a potentially new stationary object is found. Additional conditions for the pixel to be a new stationary object are that the BDG was newly created and it remains as a BDG for a long duration. Its background weight ω must be over 0.5 so that dynamic background regions are not falsely identified.

Porikli's work uses two foreground and background maps, extracted from 3D multivariate Gaussian mixture models, to find stationary objects [34]. They are computed by sampling the input data at different frame rates. Four conditions, based on the map information, are used to delineate between short-term and long-term content. The four conditions are listed as follows:

1. A pixel that is classified as both long-term and short-term foreground is considered a moving object.
2. A pixel that is classified as only long-term foreground is considered as a potential stationary object.
3. A pixel that is classified as only short-term foreground is considered an un-occluded background.

4. A pixel that is neither long-term nor short-term foreground is considered long and short-term background.

To resolve ambiguity and ascertain truly stationary objects, data from the different sampling rates are combined to remove noise and specify the time that must pass before an object can be declared as abandoned.

Jodoin et al.'s work compares activity in the current image frame to that of a *behavior image* to detect abnormal content [27]. A training period over a time window M is used to initialize the behavior image. The behavior image keeps a temporal record of information in the form of a 2D binary field. At each frame the pixel is marked with either a "one", indicating a moving object, or a "zero", indicating a static location. A large number of zeros in the binary field indicates a low activity area. An analogous window is used to measure the activity of recent image frames to create the *observed behavior image*. The behavior images are created by taking the temporal average of all the values in each pixel field. Higher numbers indicate areas of activity. For the training image the average is taken over the entire sequence length M. The observed behavior image uses a smaller window W, where $W << M$. The threshold for determining if a behavior is abnormal is based on the $\lfloor a \rfloor_0$ distance function, where a pixel is defined as abnormal if the distance between the observed behavior image and the trained behavior is above the threshold ε. The trained behavior image is adapted through a weighted average of the observed and trained behavior images. Their method was shown to work well in experiments designed to observe busing highways and highly-trafficked pedestrian intersections to detect abandoned or lingering objects.

In this chapter our approach differs from others by focusing on a storage and computational model for midground object detection that leverages the temporal record-keeping native to the multimodal mean background modeling algorithm, which is explored in Chapter 2. The midground model is designed to detect persistent stationary objects within the temporal midground window defined by the user. The user explicitly controls the detection of midground by adjusting parameters related to occlusion tolerance, object age, and temporal window.

Midground Model

The midground model builds upon the foundation of multimodal mean to allow for characterization of saliency at different timescales, while remaining efficient for implementation on resource-constrained embedded devices. The multimodal mean algorithm maintains an average value for colors that are observed at a particular pixel location. Each image pixel value is represented as a three-component color representation space (e.g., RGB, HIS). Where $I_{t,j}$ represents the color component j of a pixel in frame t (e.g., $I_{t,r}$ denotes the red component of I_t). The background model for a given pixel maintains a set of mean pixel representations, called cells. The midground model uses a modified version of the multimodal mean cell structure described in Figure 3 of Chapter 2. The midground cell structure is shown in Figure 28.

$S_{i,t,r}$	$S_{i,t,g}$	$S_{i,t,b}$	$C_{i,t}$	$OC_{i,t}$	Bday

Figure 28: Midground cell structure

Each cell ($Cell_{i,t}$) contains three mean color component values $\mu_{i,t,j}$ that have been computed over t frames for each color component x. The $S_{i,t}$ fields contain running sums of each color component, the $C_{i,t}$ field holds the decimated count while $OC_{i,t}$ holds the

observation count, and Bday is the frame number at which the cell was created. During processing, pixels in the current frame that do not match existing cells initiate the creation of a new "seed" cell. When a pixel I_t does not match a cell during processing, if either a seed cell does not exist or existing seed cells have a count less than the cell threshold *Cth*, then a new seed cell is created (the existing low-count seed cell is replaced). The seed cell's running sums are initialized to the color components of I_t, the count and observation count are initialized to one, and the birthday is initialized to *t*. Once a seed cell is matched *Cth* or more times, it becomes a regular cell and any subsequent pixel that does not match a cell will create a new seed cell. (In our experiments, *Cth* = 4.)

Multiple cells can exist at any pixel location, depending on the multimodal nature of the pixel. Based on the birthday an observation count $OC_{i,t}$, the age of a lingering object can be determined, as shown in Equation 10.

$$Age_{i,t-1} = t - B_t \tag{10}$$

Lingering and stationary objects will be occluded multiple times, given that this class of surveillance applications is often applied in public places. The midground model allows the user to set an occlusion tolerance percentage. This comes in the form of the observation density of the cell:

$$OD_{i,t} = \frac{OC_{i,t}}{Age_{i,t}} \tag{11}$$

Each cell's means are represented as a running sum for each color component $S_{i,t,x}$ an observation count, and decimated count. $S_{i,t,j}$ and $C_{i,t}$ are periodically decimated to facilitate long-term adaptation of the model while $OC_{i,t}$ remain unmodified. If the value

of the cell's decimated count $C_{i,t}$ is decimated below the *Cth*, the cell is removed. This cleans up cells corresponding to previous midground or background elements that have disappeared from the scene. (In our experiments, $d = 100$ frames/decimation.) All fields are represented as 32-bit integers. P = number of sets (pixels), K = average cells per set, M = bytes per cell, and I = average instructions per cell per frame. The storage requirement for the midground model is given as:

$$storage_{MG} = P \times K \times M \tag{12}$$

Given the average image resolution P = 400,000, with K = 1.87 cells/set average, and M = 24 bytes per cell (six words) the representation storage requirement is 18 Mbytes. The computational requirement for the midground model is defined as:

$$computation_{MG} = P \times K \times I \times fps \tag{13}$$

A more precise expression includes mid-set matching effects and decimation costs. However, since sets are usually small and decimation is infrequent, this model is a good approximation. For the experiment conducted in this paper at one frame per second (fps), the computational requirement is approximately 15 Mops/second. While other processes (image extraction, preprocessing, and post-midground processing) are not considered, the midground model computation and storage is well within the capabilities of today's embedded processors.

Experiments and Results

Four test sequences are used to evaluate the midground detection technique, as shown in Figure 29.

Figure 29: Test sequences with midground regions highlighted

The first sequence, “Blocks”, is an idealized test set in which blocks are added and removed from a uniformly illuminated scene with no foreground activity or dynamic background. In this sequence two background blocks are removed and a new block is added prior to the evaluation frame. Only one true midground object should appear in the scene. In the ‘Outdoors’ sequence, a busy outdoor scene includes dynamic background (waving trees, sky), many foreground objects, and two true midground objects (a trash can and a small paper bag) that are frequently occluded. “PETS” 2006 dataset S7 [32] includes a train terminal where a small bag is left unattended. i-Lids (AVSS) [23] is a tube (subway) station where a large bag is placed. In this pixel-level evaluation, no attempt is made to determine whether the bag is unattended. Only a cell’s age and observation density are used to determine its presence in midground. However, the information produced through midground detection can be extended to use with a suitable proximity algorithm to detect more complex events.

Our experiments specifically evaluate the quality of midground detection when varying the observation density threshold (which affects tolerance to occlusion) and

frame rate. Table 20 provides the sequence length (in seconds), frame rates, midground time window (in seconds), and number of midground objects.

Table 20: Midground test sequences summary

Sequence	Duration (minutes)	fps	MG Window (seconds)	MG objs
Blocks	13.3	1	200 to 250	1
Outdoors	16.7	1	200 to 250	2
PETS	2.3	1 to 25	20 to 40	1
AVSS i-Lids	2.0	1 to 25	20 to 40	1

For the high frame rate PETS and AVSS sequences, the frames were downsampled to 1, 2, 4, 8, 16, and 25 frames per second.

Varying observation density threshold

Dense and heavily trafficked scenes will result in the occlusion of midground objects. To ensure detection, the model permits a user-defined level of occlusion tolerance, as specified by the observation density threshold. The observation density is a ratio of the age of the object and the number of times it has been seen. If a pixel's observation density is above the specified threshold, it will be declared midground. The effect of varying observation density threshold, at a sequence frame rate of 1 fps, is shown in Figure 30.

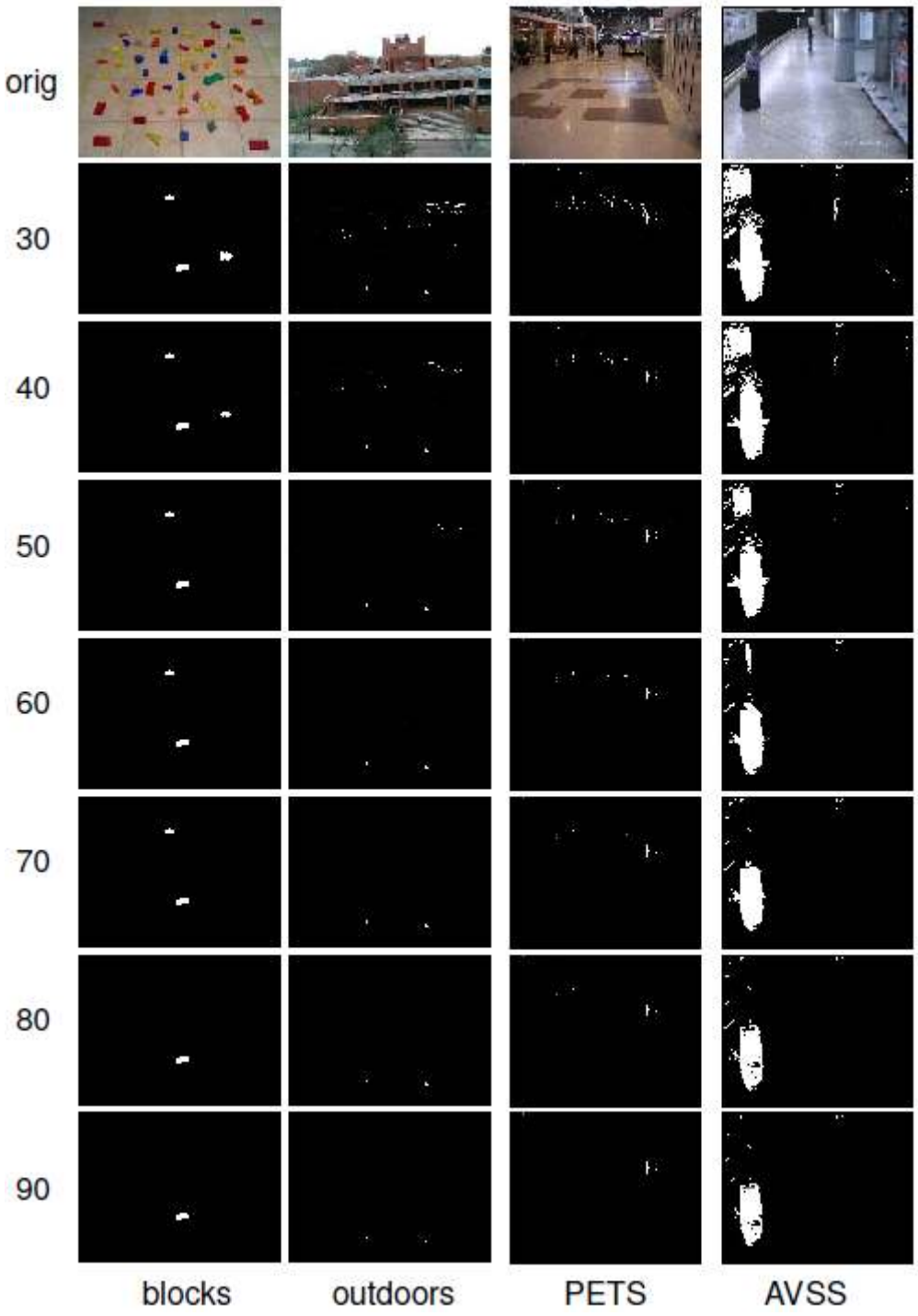

Figure 30: Effects of observation density threshold

For each sequence the threshold is varied from 30% – 90%. The lower density thresholds contain greater midground false positives, resulting primarily from transient cell values of objects that have left the scene. The lower density threshold relaxes the criterion for

midground detection, making it possible for heavily occluded objects to be picked up. However, it has the advantage of potentially picking up scene noise as midground. In the "Blocks" sequence, the removed blocks appear in the midground at lower T_{OD} values since their corresponding cells satisfy the midground age requirements. At higher T_{OD}, they are excluded. A similar reduction in false positive noise is exhibited in the other sequences until 80%, above which false negatives begin to appear within the midground objects.

Since the MG window is an independent parameter, it can be adjusted to precisely meet the application requirements. In "Outdoors", the window has been evaluated over relatively long windows (tens of minutes) whereas the transportation sequences typically require shorter windows (tens of seconds). Since the representation does not explicitly include the temporal window or T_{OD}, the evaluation function can include additional factors in midground determination. For example, human proximity information combined with a bag's midground representation to trigger an alarm.

Varying sampling frame rate

The impact of varying frame rate was explored in a second experiment, with the results shown in Figure 31.

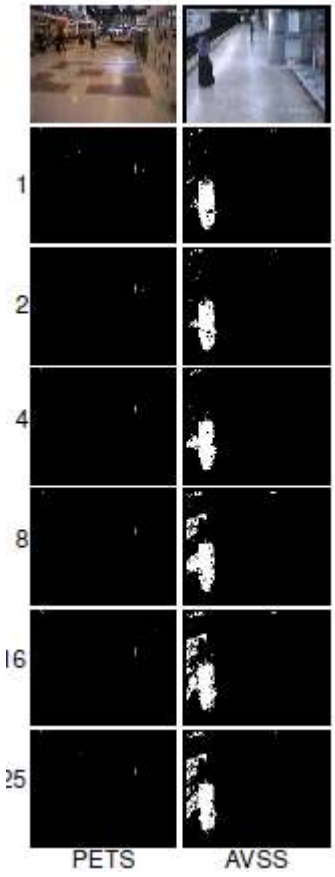

Figure 31: Effects of varying frame rate

In this set of experiments, midground (defined with a constant temporal window and T_{OD} = 80%) is evaluated at frame rates of 1, 2, 4, 8, 16, and 25 fps. While the frame rate appears to have little effect on the PETS sequence, false positive noise is significantly reduced in the AVSS sequence as frame rate is reduced. Normally, higher frame rates reduce the impact of random image noise through averaging. In this sequence, the higher frame rates created more permanent objects (by exceeding T_C). Since computation is linearly proportional to frames processed, it is highly desirable to operate at the lowest possible frame rate for a given application and scene.

Computational Requirements

For the experiments conducted, with a sequence sampling of one frame per second (fps), the computational requirement is approximately 15 Mops/second. While other processes (image extraction, preprocessing, and post-midground processing) are not considered, the midground model computation and storage is well within the capabilities of today's embedded processors. For a 400,000 pixel image at one fps, the algorithm requires only 18 Mbytes storage and 15 Mops/sec processing throughput. Using realistic test sequences, this technique is shown to detect midground objects with high accuracy. These tests show that a high T_{OD} (>80%) can be used to minimize false positives. The technique performs well at relatively low frame rates (~1 fps) suggesting that it can be applied to a downsampled image stream for greater efficiency. This midground detection technique can serve as a valuable complement to more complex scene analysis and event detection methods. Its low cost and high accuracy can provide scene modeling in addition to traditional foreground and background information.

CHAPTER 5 : CONCLUSION AND FUTURE WORK

This dissertation explores the development of temporal scene analysis techniques and optimization strategies for implementation of early vision algorithms on embedded platforms. We perform our experiments on real-world scenes, comprising a mix of industry created test sequences as well as ones that were specifically captured for this dissertation using commercial-off-the-shelf webcams. To evaluate the run-time performance of our optimization techniques, we use an embedded platform, the eBox-2300.

The first contribution defines a fast adaptive, multimodal, and storage efficient background modeling algorithm, called *multimodal mean*. The effects of thresholding on computational complexity and accuracy, and the effects of adaptation rate on long-term adaptation and temporal record-keeping are explored. The second contribution introduces an optimization technique to improve the run-time performance of pixel-based background models on embedded platforms. It uses a color clustering technique to identify stable, large, and permanent background objects in the scene to process them quickly, improving run-time performance while maintaining comparable accuracy. In the third contribution a technique for temporal analysis, called *midground*, is introduced. This method extends scene analysis from a bi-state background/foreground detection framework to one including the temporal realms of foreground, midground, and background. This technique provides a set of user-tunable temporal parameters that can be applied to applications such as abandoned object detection, illegally parked vehicle detection, and loitering detection

Summary of Contributions and Results

5.1.1 Multimodal mean background modeling technique:

- Evaluated the effect of thresholding techniques and adaptation rates on the accuracy and compute load of MMean:
  - Fixed thresholding on MMean compares favorably to standard deviation based thresholds in scenes with moderate dynamic backgrounds.
  - MMean can precisely control the length of object persistence under occlusion in scenes with stable environmental conditions.
  - The multimodal mean, when mapped to a 200 MHz platform, runs 6.2x faster than MoG with 18% less storage.

5.1.2 Chromatic distribution analysis for optimized background modeling on embedded platforms:

- Introduced a novel color clustering technique to identify large stable and permanent background objects for suppressed adaptation and faster processing
  - This technique reduces storage operations by 58%.
  - It provides a 45% performance improvement on the 200MHz eBox-2300.

5.1.3 Midground model for temporal scene analysis:

- Presented a fast and storage efficient model for detection of stationary objects
  - Model needs only 18MB for storage and 15Mops/sec for processing.

- It performs accurately at relatively low data sampling rate (1 frame sampled per second).

Future Work

Future work will investigate the use of the color clustering technique described in Chapter 3 for adaptation and compensation of lighting changes. Since the locations of large, permanent objects are known in the scene, this information can be used to distinguish changes in color between those caused by occlusions and those caused by lighting. This information can be used to improve the foreground detection and long-term adaptation accuracy of the multimodal mean and midground algorithms. Evaluation of this work would not be limited to midground and multimodal mean. It has the potential to be used in a number of different algorithms, since lighting variations continue to be a challenge in the computer vision field.

REFERENCES

[1] S. Apewokin, B. Valentine, S. Wills, L. Wills, A. Gentile, "Multimodal Mean Adaptive Backgrounding for Embedded Real-Time Video Surveillance," *Embedded Computer Vision Workshop (ECVW07)*, June 2007.

[2] S. Apewokin, B. Valentine, R. Bales, S. Wills, L. Wills, "Tracking Multiple Pedestrians in Real-Time Using Kinematics," *IEEE Embedded Computer Vision Workshop (ECVW08)*, June 2007.

[3] S. Apewokin, B. Valentine, J. Choi, S. Wills, L. Wills, "Real-Time Adaptive Background Modeling for Multicore Embedded Systems", *The Journal of Signal Processing Systems,* Springer 2008.

[4] S. Apewokin, B. Valentine, D. Forsthoefel, S. Wills, L. Wills, A. Gentile, "Embedded Real-Time Surveillance Using Multimodal Mean Background Modeling," in *Embedded Computer Vision*, B. Kisacanin, S. Bhattacharyya, S. Chai, (editors), Springer, 2009.

[5] S. Apewokin, B. Valentine, S. Wills, L. Wills, "Cat-Tail; DMA: Efficient Image Data Transport for Multicore Embedded Mobile Systems," *Journal of Mobile Multimedia, Special Issue on Recent Advances in Mobile and Multimedia Applications,* Accepted January 2010.

[6] K. Appiah, A. Hunter, "A single-chip FPGA implementation of real-time adaptive background model," *Image and Vision Computing*, 2009.

[7] T.J. Atherton, D.J. Kerbyson, "Reducing false alarm rates in surveillance imaging using significance testing," *IEE Colloquium on Image Processing for Security Applications*, pp. 7/1-7/4, March 1997.

[8] S. Atev, O. Masoud, N. Papanikolopoulos, "Practical Mixtures of Gaussians with Brightness Monitoring," *IEEE Intelligent Transportation Systems Conference*, pp. 423-428, 2004.

[9] S. Bhandarkar, X. Luo, "Fast and Robust Background Updating for Real-time Traffic Surveillance Monitoring," *IEEE Conference on Computer Vision and Pattern Recognition,* 2005.

[10] Z. Chaohui, D. Xiaohui, X. Shuoyu, S. Zheng, and L. Min, "An Improved Moving Object Detection Based on Frame Difference and Edge Detection," *Forth International Conference on Image and Graphics*, pp. 519-523, IEEE Computer Society, 2007.

[11] S. Cheung, C. Kamath, "Robust techniques for background subtraction in urban traffic video," *Video Communications and Image Processing*, SPIE Electronic Imaging, (2004) 881-892.

[12] J.W. Choi, S. Apewokin, B. Valentine, D. Wills, L. Wills, "Edge noise removal in multimodal background modeling techniques," SPIE 2008.

[13] R. Cucchiara, C. Grana, M. Piccardi, A. Prati, "Detecting Moving Objects, Ghosts, and Shadows in Video Streams", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25: (11), October 2003.

[14] P. Dickinson, A. Hunter, K. Appiah, "A spatially distributed model for foreground segmentation", IEEE International Conference on Field-Programmable Technology, pp. 95-102, 2005.

[15] P. Dickinson, A. Hunter, K. Appiah, "A spatially distributed model for foreground segmentation", Journal of Image and Vision Computing, pp. 1326-1335, (27) (9) August 2009.

[16] "DMP Electronics Inc.: VESA PC eBox-2300 Users Manual" (Available online, 2006), http://www.wdlsystems.com/downloads/manuals/1EBOX23 m.pdf.

[17] A. Elgammal, D. Harwood, and L. Davis, "Non-Parametric Model for Background Subtraction," *in Proceedings of IEEE ICCV'99 Frame-rate workshop*, September 1999.

[18] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, 1972.

[19] M. Gervautz, W. Purgtathofer, "A simple method for color quantization: octree quantization," *Graphics Gems*, Academic Press, 1990, pp. 287-293.

[20] A. Hampapur, L. Brown, J. Connell, S. Pankanti, A. Senior, Y. Tian, Smart Surveillance: Applications, Technologies and Implications, IBM T.J. Watson Research Center, (Available online, 2009), http://www.research.ibm.com/peoplevision/PCM03.PDF.

[21] A. Hampapur, R. Bobbitt, L. Brown, M. Desimone, R. Feris, R. Kjeldsen, M. Lu, C. Mercier, C. Milite, S. Russo, C. Shu, Y. Zhai, "Video Analytics in Urban Environments", *IEEE International Conference on Advanced Video and Signal Based Surveillance*, 2009.

[22] P. Heckbert, "Color Image Quantization for Frame Buffer Display," *ACM Comput. Graph.*, (17) 3, (1982) 297-307.

[23] i-Lids dataset for AVSS 2007, (Available online, 2007) www.elec.qmul.ac.uk/staffinfo/andrea/avss2007 d.html.

[24] N. Jojic, A. Perina, M. Cristani, V. Murino, and B. Frey, "Stel component analysis: Modeling spatial correlations in image class structure," in *Proc. of Computer Vision and Pattern Recognition Workshops*, June 2009, pp. 2044 - 2051.

[25] A. Jain, M. Murty, P. Flynn, "Data clustering: a review," *ACM Computing Surveys*, 3 (31) (1999) 264-323.

[26] S. Jianbo, J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22 (8) (2000) 888-905.

[27] P. Jodoin, J. Konrad, V. Saligram, "Modeling Background Activity for Behavior Subtraction," *IEEE International Conference on Distributed Smart Cameras,* September 2008.

[28] K. Kiratiratanapruk, P. Dubey, S. Siddhichai, "A Gradient-Based Foreground Detection Technique for Object Tracking in a Traffic Monitoring System," *Proceedings of the IEEE International Conference on Advanced Video and Signal Based Surveillance*, 2005, pp. 377-381.

[29] J.B. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations," *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, Berkeley, University of California Press, (1) 1967, pp. 281-297.

[30] R. Mathew, Z. Yu, and J. Zhang, "Detecting New Stable Objects in Surveillance Video," *IEEE Workshop on Multimedia Signal Processing*, pp. 1- 4, October 2005.

[31] N. McFarlane, and C. Schofield, "Segmentation and tracking of piglets in images," *Machine Vision and Applications*, 8(3), pp. 187–193, 1995.

[32] PETS 2006 dataset, Sequence Name S7-T6-B: Video 1, (Available online, 2007), http://www.cvg.rdg.ac.uk/PETS2006/data.html

[33] M. Piccardi, "Background subtraction techniques: a review," *IEEE International Conference on Systems, Man and Cybernetics*, Vol. 4, pp. 3099-3104, October 2004.

[34] F. Porikli, "Detection of Temporarily Static Regions by Processing Video at Different Frame Rates," *Proceedings of the IEEE Conference on Advanced Video and Signal Based Surveillance*, 2007, pp. 236-241.

[35] R.J. Radke, S. Andra, O. Al-Kofahi, B. Roysam, "Image Change Detection Algorithms: A Systematic Survey," *IEEE Transactions on Image Processing*, 14(3): pp. 294-307, March 2005.

[36] P. Rosin, E. Ioannidis, "Evaluation of global image thresholding for change detection, *Pattern Recognition Letters* 24 (15) (2003) 332-343.

[37] G. Sharma, H.J. Trussell, "Digital Color Imaging," *IEEE Transactions on Image Processing*, 6 (7) (1997) 901-932.

[38] P. Sneath, R. Sokal, "Numerical Taxonomy. The principle and practice of numerical classification", W.H. Freeman, 1973.

[39] P. Spagnolo, A. Caroppo, M. Leo, T. Martiriggiano, T. D'Orazio, "An Abandoned/Removed Objects Detection Algorithm and Its Evaluation on PETS Datasets," *IEEE International Conference on Advanced Video and Signal Based Surveillance*, November 2006.

[40] C. Stauffer, and W.E.L. Grimson, "Learning Patterns of Activity Using Real-Time Tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8), pp. 747-757, August 2000.

[41] K. Toyama, J. Krumm, B. Brummitt, and B. Meyers, "Wallflower: Principles and Practices of Background Maintenance," *in Proc. of ICCV* (1), pp. 255-261, 1999.

[42] B. Valentine, J. Choi, S. Apewokin, L. Wills, S. Wills, "Bypassing BigBackground: An Efficient Background Model for Embedded Video Surveillance," *Proceedings of the IEEE International Conference on Distributed Smart Cameras*, 2008.

[43] B. Valentine, S. Apewokin, L. Wills, S. Wills, "An Efficient, Chromatic Clustering-Based Background Model for Embedded Vision Platforms", Under Review, *Computer Vision and Image Understanding, Special Issue: Embedded Vision*, Submitted Jan 2009.

[44] B. Valentine, S. Apewokin, S. Wills, L. Wills, A. Gentile, "Midground Object Detection in Real World Video Scenes," *IEEE Conference on Advanced Video and Signal Based Surveillance,* September 2007.

[45] L. Xu, "Issues in Video Analytics and Surveillance Systems: Research/Prototyping vs. Applications/User Requirements," *IEEE Conference on Advanced Video and Signal Based Surveillance,* September 2007.

[46] Q. Zang, R. Klette, "Robust Background Subtraction and Maintenance," *17th IEEE International Conference on Pattern Recognition*, 2004.

[47] Texas Instruments, "TMS320C6474 Multicore Digital Signal Processor" http://focus.ti.com/lit/ds/symlink/tms320c6474.pdf (Available online, 2009)

[48] Analog Devices, "Blackfin Embedded Symmetric Multiprocessor" http://www.analog.com/static/imported-files/data_sheets/ADSP-BF561.pdf (Accessed 12/2009).

[49] ARM, "Cortex Family" http://www.arm.com/products/CPUs/families/CortexFamily.html (Available online, 2009)